Kalle Kuronen

Lean software development – Kanban-prosessimalli ohjelmistokehitystyössä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

5.5.2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Kalle Kuronen Lean software development – Kanban-prosessimalli ohjelmistokehitystyössä 43 sivua 05.05.2014
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja(t)	yliopettaja Erja Nikunen yliopettaja Auvo Häkkinen

Insinöörityön tavoitteena oli selvittää, onko Lean-ajattelumallista ja sitä toteuttavasta Kanban-prosessimallista hyötyä ohjelmistokehitystyössä. Tavoite oli tarkoitus saavuttaa tutkimalla kirjallista aineistoa Kanbanin tai Leanin käytöstä erilaisissa organisaatioissa sekä haastatella näitä malleja hyödyntävien organisaatioiden edustajia.

Kirjallisen aineiston ja haastattelujen avulla tavoitteena oli myös selvittää, mitä mahdollisia hyötyjä ja haittoja Lean-mallista on suhteessa muihin ketteriin ohjelmistokehitysmenetelmiin, ensisijaisesti Scrumiin.

Kirjallisuusselvityksen ja haastattelujen lisäksi tavoitteena oli ottaa Kanban käyttöön yhdessä Toinen veli Oy:n pilottiprojektissa ja tutkia, kuinka helppoa tai vaikeaa käyttöönotto on.

Kirjallista aineistoa haettiin sellaisten ohjelmistokehitystä tekevien organisaatioiden ympäriltä, jotka olivat siirtyneet käyttämään Kanbania tai jotka käyttivät ylipäätään ketteriä ohjelmistokehitysmenetelmiä. Kanbanin käyttöönottoprojektiksi sovittiin tilaajayritys Toinen veli Oy:n KiinteistöVELI-tuotteen ympärille perustettava blogi.

Kirjallinen aineisto osoitti tutkittujen menetelmien käyttöönoton tuovan merkittäviä hyötyjä ohjelmistokehitykseen. Kanbanin käyttö lyhensi ohjelmistotuotteiden toimitusaikaa, nosti ohjelmistojen laatua, helpotti kommunikointia ja yhteistyötä, nosti toimitusvarmuutta ja vähensi asiakkaiden raportoimia ohjelmistovirheitä.

Käyttöönottoprojekti vahvisti kirjallisen aineiston väittämät nopeasta käyttöönotosta ja todisti myös läpimenoajan lyhentyneen projektin edetessä.

Työ osoitti, että vaikka Kanbanin ja Leanin käytöstä ohjelmistokehitystyössä löytyy niukasti tutkittua tietoa ja selkeitä malleja, ovat menetelmien käytöstä saadut tulokset rohkaisevia ja mallien käyttöönotto helppoa, jos niiden taustalla olevat ajatukset ymmärretään oikein.

Avainsanat	Lean, Kanban, Scrumban, ketterä ohjelmistokehitys, virtaviivainen ohjelmistokehitys

Author(s) Title Number of Pages Date	Kalle Kuronen Lean Software Development – Utilization of Kanban Process Model in Software Engineering 43 pages 5 May 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Erja Nikunen, Principal Lecturer Auvo Häkkinen, Principal Lecturer

The aim of this study was to explore the Lean philosophy and the Kanban process model to determine if these philosophies and Kanban implementing them would be beneficial for usage in software engineering. This aim was achieved by inspecting literary studies and by interviewing people from organizations utilizing these models.

Another aim of this study was to examine the possible positive or negative effects of Lean methodologies compared to other agile software development methods and more specifically to Scrum. In addition to literary examination also a pilot project was executed to inspect how simple or complex deploying the Kanban process model is.

The literary research was targeted on studies focusing on organizations that have deployed Kanban in their software development or to organizations utilizing agile software development methods.

The pilot project for studying the ease of deployment of Kanban consisted of building a blog.

The literary research revealed that the adoption of Lean methodologies lead to significant benefits in software engineering. The Kanban method usage improved the lead time to deliver software, improved the quality of software, improved communication and coordination, increased consistency of delivery and decreased the number of defects reported by customers.

Deployment of the pilot project confirmed the statements made in the literature about easy adoption and also proved that the lead time was reduced during the project.

The study revealed that there are currently not enough studies about using Kanban and Lean philosophies in software engineering. Nevertheless, the results are encouraging and the adoption of these models is easy if the underlying principles beneath them are understood.

Keywords	Lean, Kanban, Scrumban, Agile software development, Lean software development

Sisällys

Lyhenteet

Blogi	Viittaa verkkosivuun, jossa julkaistaan kirjoitettuja verkkokirjoituksia.
Bugi	Ohjelmointivirhe on ohjelmiston lähdekoodissa oleva virhe.
JIT	Just-in-Time viittaa johtamisfilosofiaan, jossa ajatus on toimittaa ainoastaan tuotteita niitä tarvitsevalle asiakkaalle juuri silloin, kun niitä tarvitaan, ja juuri oikea määrä.
JOT	Juuri Oikeaan Aikaan. Suomenkielinen lyhennevastine JIT-lyhenteelle.
LSD	Lean Software Development. Viittaa ajattelumalliin tai johtamisfilosofiaan, jonka ydin on puhdasta arvoa tuottava tuotanto.
SF	Software Factory. Viittaa Helsingin yliopiston tietojenkäsittelytieteen laitoksen alaiseen laboratorioon, jossa opiskelijat kehittävät asiakastilauksista ohjelmistotuotteita.
TDD	Test-Driven Development. Tarkoittaa ohjelmistokehitysmallia, jossa ohjelmiston testitapaukset tehdään ennen varsinaista ohjelmakoodia.
TPS	Toyota Production System on autovalmistaja Toyotan luoma ja käyttämä Lean-ajattelua toteuttava tuotantomalli.
WIP	Work-in-Progress on Kanban-prosessimallin vaatima samanaikaisten tehtävien määrärajoitus.

1 Johdanto

Tässä työssä tutkitaan Lean-ajattelumallia yleisesti ohjelmistokehitystyön näkökulmasta. Aiemmin alun perin Toyotassa käyttöönotettua Kanbania on hyödynnetty tuotannollisissa prosesseissa kuten tuotantolinjoilla, mutta nykyään sitä käytetään enenevissä määrin myös ohjelmistokehityksessä. Työssä painotetaan, etenkin kirjoitushetkellä jonkinasteista nostetta nauttivan, Kanbanin soveltumista ohjelmistokehitysprojekteihin. Työssä pyritään syventymään Lean-prosessimallien ajatusmaailmaan, sen sisältöihin, ideologioihin ja sen tarjoamiin mahdollisuuksiin. Lisäksi työssä pohditaan, nopeuttaako, helpottaako tai tehostaako Lean-prosessimallin käyttö ohjelmistoprojektin lopullista tulosta, joko laadullisesti tai resurssillisesti. Myös Lean-ajattelumallin tuottaman ohjelmiston ylläpidettävyyttä ja aikataulujen pitävyyttä arvioidaan ja selvitetään sitä, vähentääkö se virheitä lopullisessa tuotteessa.

Työn tavoitteena on selvittää, onko Leanin käyttäminen edellä mainitut seikat huomioiden järkevää ohjelmistoprojekteissa ja jos, niin minkälaisissa projekteissa. Kanbanin käyttöönottoa kokeillaan, ja tarkoituksena on kuvata sen mahdollisia hyötyjä ja haasteita. Leanin hyötyjä ja haittoja arvioidaan suhteessa muihin vallalla oleviin ketteriin ohjelmistokehitysmalleihin, ensisijaisesti Scrumiin.

Työn tulokset johdetaan paitsi kirjallisista lähteistä myös käytännön ohjelmistokehitysprojektista, joka tehdään tai pyritään tekemään Kanbanin avulla Lean-ajattelumallin mukaisesti.

Kokeellinen projektityöskentely tämän insinöörityön osalta on vähäistä ja koskettaa vain yhtä projektia, joka sekin on tiimin ensimmäinen Kanbanilla suorittama projekti. Tieteellisesti täsmällisiä johtopäätöksiä Kanbanin hyödyistä tai haitoista ei voi pelkästään sen perusteella tehdä. Työssä pyritään myös tuomaan muiden Kanbania käyttävien organisaatioiden näkemyksiä ja kokemuksia esiin.

2 Ketterät ohjelmistokehitysmenetelmät

Ohjelmistotuotteet poikkeavat radikaalisti konkreettisemmista tuotteista tai projekteista. Jos esimerkiksi saneerataan keittiö, voidaan sinne valita tietyistä materiaalivaihtoehdoista ja kodinkoneista itselleen sopivat tuotteet. Valittavaksi tulevat esimerkiksi lattia-

materiaali, työtasot, keittiökalusteet ja laitteet maun mukaan. Ohjelmistotuotteita kehitettäessä on hyvin mahdollista ja jopa todennäköistä, että projektin alkaessa ei ole olemassa olevaa ratkaisua, jonka voisi valita. On mahdollista, että valittavaksi tulee ratkaisu, joka on ensin projektissa täytynyt luoda ja kehittää.

Ketterä ohjelmistokehitys sisältää vesiputousmallin kanssa samankaltaisia työvaiheita, mutta työvaiheiden suoritusjärjestys ja sisältö sekä sen laajuus poikkeavat vesiputousmallista. Myös itse kehitysprosessi eroaa vesiputousmallista. Ketterässä ohjelmistokehityksessä projektin aikana syntyvää tietoa pystytään joustavammin hyödyntämään ja siksi se onkin kasvattanut suosiotaan suhteessa vesiputousmalliin (Brendan ym. 2013).

2.1 Ketterän kehityksen määritelmä

Ketterä ohjelmistokehitys nojaa ketterän ohjelmistokehityksen julistukseen (Beck, ym. 2001) ja sen taustalla oleviin periaatteisiin. Julistus sanoo ketterän ohjelmistokehityksen arvostavan

- yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja
- toimivaa ohjelmistoa enemmän kuin kattavaa dokumentointia
- asiakasyhteistyötä enemmän kuin sopimusneuvotteluja
- vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa.

Käytännössä ketterällä ohjelmistokehityksellä tarkoitetaan sellaista ohjelmistokehitystä, jossa ohjelmistoa kehitetään iteratiivisesti ja inkrementaalisesti. Iteratiivinen tässä kontekstissa tarkoittaa sitä, että ohjelmistokehitystä tehdään toistuvissa (*iteratiivisissa*) sykleissä, jotka koostuvat suunnittelusta, vaatimusmäärittelystä, suunnittelusta ja toteutuksesta, implementoinnista (*käyttöönotosta*) ja testauksesta sekä evaluoinnista. Näitä toistuvia syklejä voidaan kutsua tutummin iteraatioiksi. Inkrementaalisella (*eng. incremental*) käsitetään ohjelmiston kehityksessä pienten osakokonaisuuksien toteuttamista kerrallaan.

Jokaisen iteraation lopussa saavutetut tulokset käydään läpi ja seuraavan iteraation käynnistyspalaverissa päätetään, mitä seuraavaksi tehdään. Koska tulokset voivat muuttua lähtökohdan tilanteista, on luonnollista tehdä päätös seuraavaksi tehtävistä

asioista vasta edellisen iteraation päätyttyä. Siinä vaiheessa organisaatio on oppinut ja pystyy tekemään parempia päätöksiä uuteen tietoon pohjautuen.

Sanana ketterä tässä kontekstissa tulee siitä, että em. tavoin toimiessa muutoksiin voidaan reagoida hyvin nopeasti, eli *ketterästi*.

Myös Lean-ajattelun tai vähintään Kanbanin voidaan katsoa olevan ketterää ohjelmistokehitystä (Fichtner 2012). Vaikka Kanbanissa ei lähtökohtaisesti iteraatioita olekaan, niin kehitystyössä noudatetaan pitkälti samoja arvoja kuin mitä ketterän ohjelmistokehityksen julistus sanoo.

2.2 Syntyhistoria ja taustat

Ketterä ohjelmistokehitys on vasta viimeisen vuosikymmenen aikana saavuttanut suuren suosion (Brendan ym. 2013), vaikka jo vuonna 1957 IBM:llä työskennellyt Gerald M. Weinberg kertoo muistelmissaan (Mitchell 2013) heidän toteuttaneen inkrementaalista ohjelmistokehitysmallia. Hän myös kertoo heidän kokeneen jo tuolloin, että vesiputousmalli oli isoissa projekteissa vajavainen, tai vähintäänkin ohjelmistokehitystyön realiteeteista piittaamaton tapa työskennellä. Ensimmäinen tieteellinen artikkeli aiheesta löytyy vuodelta 1968, jossa varsinaisesti käytetään sanaparia iteratiivinen kehitys.

2.3 Vesiputousmalli ja virheellinen käsitys

Vesiputousmalli on vaiheellinen ohjelmistotuotantoprosessi, joka eroaa oleellisesti ketterästä ohjelmistokehityksestä. Mallissa painotetaan jokaisen vaiheen virheettömyyttä eikä vaiheen jälkeen ole mahdollista palata edellisiin vaiheisiin. Vesiputousmallissa kehitysprosessi etenee suunnittelusta toteutuksen läpi aina ylläpitoon asti vesiputouksen tavoin.

Kehitysprosessi vesiputousmallissa alkaa aina mahdollisimman tarkasta vaatimusten määrittelystä. Kun on selvitetty ja kirjattu ylös vaatimukset, aloitetaan ohjelmiston suunnittelu, joka sisältää niin ohjelmiston arkkitehtuurin suunnittelua, mutta myös konkreettisten toiminnallisuuksien suunnittelua. Jälleen suunnitteluvaiheen jälkeen siirrytään toteuttamaan suunnitteluvaiheen viitoittamalla tiellä varsinaista ohjelmistoa, eli kirjoittamaan ohjelmakoodia.

Kun toteutus, integraatio, testaus ja asennus on tehty, siirrytään ylläpitovaiheeseen, jossa ohjelmiston katsotaan olevan valmis.

Kuva 1. Vesiputousmalli (Lähde: Wikipedia)

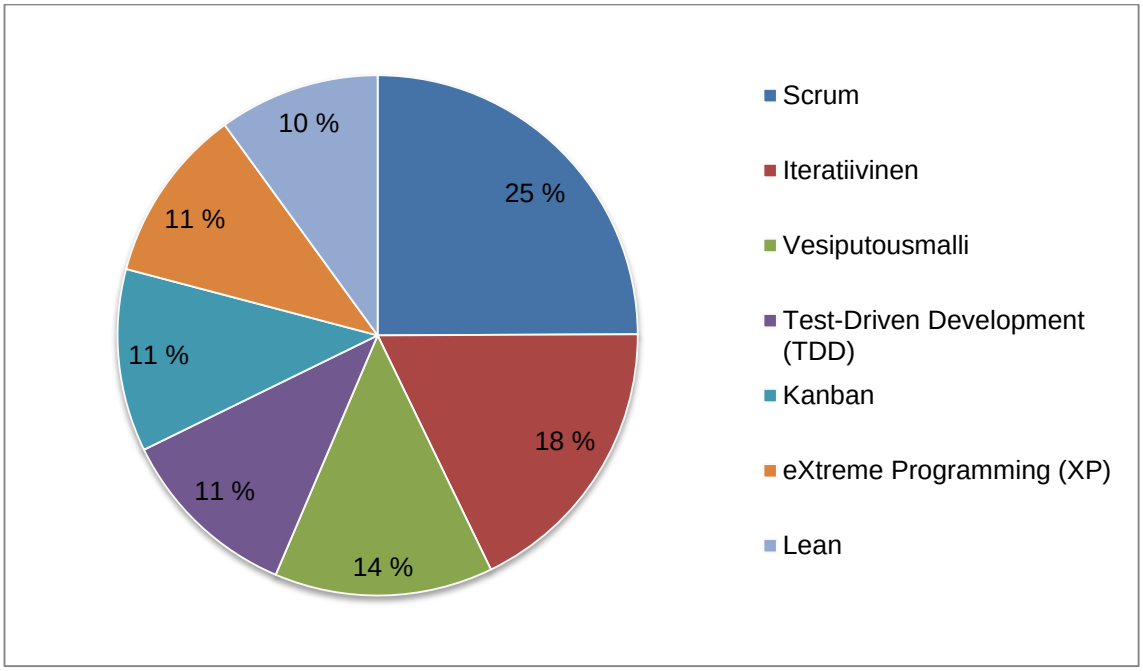
Aina ennen seuraavaan vaiheeseen siirtymistä suoritetaan katselmus, jossa tarkastellaan käynnissä olevan vaiheen tilannetta, ja päätetään, voidaanko seuraavaan vaiheeseen siirtyä vai ei. Katselmus on siksi tärkeä, että jos vesiputousmallia noudatetaan täysin oikein, ei paluumahdollisuutta edellisiin vaiheisiin ole. (Tauriainen 2005.)

Vesiputousmallista on luotu useita malleja, yhtenä niistä muunnos, joka on osittain iteratiivinen. Tässä mallissa lineaarisuudesta voidaan poiketa. Silloin jokaisen vaiheen jälkeen voidaan tarvittaessa myös palata edelliseen vaiheeseen. (Tauriainen 2005.)

Yllä esitetty kuvaus vesiputousmallista on yleisen käsityksen mukainen, vaikka mallin isänä pidetyn Winston Roycen alkuperäinen ajatus on ollut toinen. Royce ehdottikin, että uutta ohjelmistoa kehitettäessä asiakkaalle ei annettaisi ensimmäisen vesiputousmalliprojektin tuotosta, vaan ensimmäisen projektin jälkeen aloitettaisiin toinen projekti, jossa aiemmin luotua uutta ohjelmistoa jatkokehitettäisiin. Asiakas saisi vasta sen version, jota on jo jatkokehitetty. Tämä Roycen esitys jätettiin kuitenkin huomiotta, kun vesiputousmalli saavutti suuren yleisön ja niinpä sitä ei yleensä tunneta. (Larman & Basili 2003.)

2.4 Eri mallien käyttöasteet

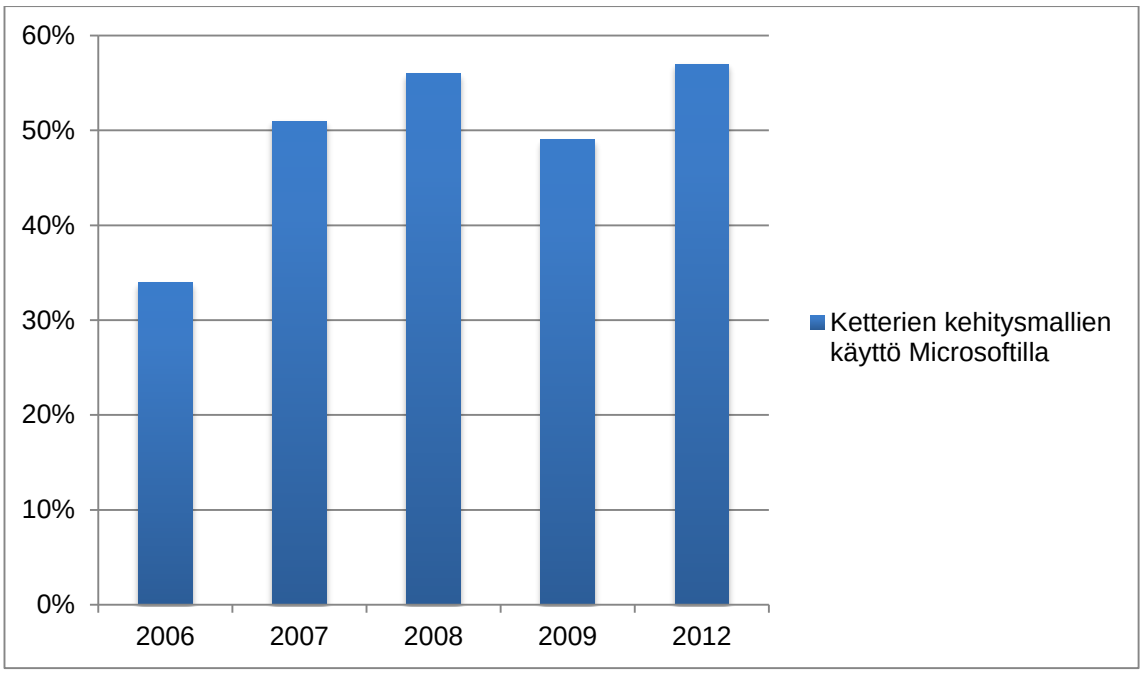
Forrester Group tutki marraskuussa 2011 kansainvälisesti ketterien ohjelmistokehitysmenetelmien käyttöä verkkokyselytutkimuksella. Kysely kohdennettiin sellaisille IT-alan ammattilaisille, jotka ovat implementoimassa tai jo implementoineet ketterät ohjelmistokehitysmenetelmät.



Kuva 2. Eri kehitysmenetelmien käyttö Forrester Groupin 2011 -kyselytutkimuksen mukaan

Kuten yllä olevasta graafista voidaan nähdä, menetelmistä käytetyin on ehdottomasti Scrum. Tulokset eivät ole toisiaan poissulkevia, joten organisaatio, joka käyttää Scrumia, kehittää tuotteitaan iteratiivisesti. Samoin TDD:tä voidaan toteuttaa eri kehitysmallien sisällä.

Microsoft julkaisi seurantatutkimuksen, joka ajoittui kuuden vuoden aikajaksolle vuodesta 2006 vuoteen 2012 ja siinä tutkittiin ketterien ohjelmistokehitysmenetelmien käyttöä Microsoftin oman organisaation sisällä. Microsoftin tekemä tutkimus on laajin ja tutkimukseen onkin saatu 1969:n henkilön otanta. Tutkimus osoittaa ketterien ohjelmistokehitysmenetelmien kasvaneen ainakin Microsoftin sisällä. (Murphy ym. 2013.)



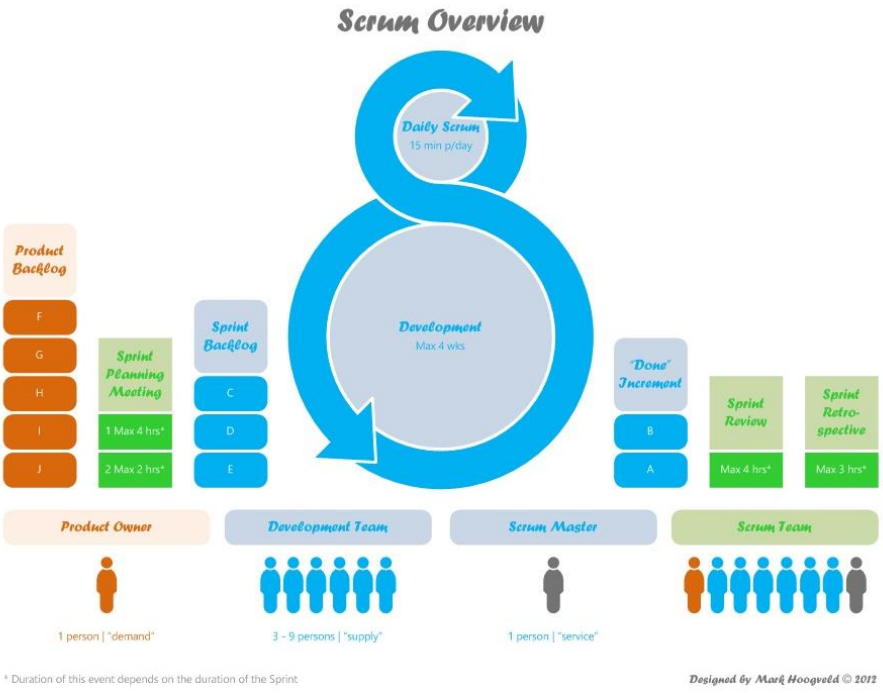
Kuva 3. Ketterien kehitysmallien käyttö Microsoftilla

Kuvasta 3 voidaan havaita, että ketterien ohjelmistokehitysmenetelmien käyttö on seurantajakson aikana keskimäärin kasvanut, ja 57 % kyselytutkimukseen vastanneista käytti ketteriä ohjelmistokehitysmenetelmiä vuonna 2012 Microsoftilla.

2.5 Scrum

Scrum on ohjelmistotuotantoprosessi ja tarjoaa projektinhallinnan viitekehyksen, joka sisältää varsin tarkat määritelmät organisaation rakenteesta, projektien ajoittamisesta ja läpiviennistä. Kaikkia Scrumin sääntöjä ei tässä esitetä, mutta sen sijaan tyypillinen Scrum-projekti ja sen pääasiat on esitetty alla.

Scrum-projektissa työskennellään inkrementaalisesti, ja Scrum-projekti koostuu iteraatioista. Inkrementaalinen työskentely ja iteraatiot on selitetty tarkemmin luvussa 2.1. Scrumissa iteraatioita kutsutaan pyrähdyksiksi (engl. *Sprint*), jotka kestävät 2-4 viikkoa, ja niiden määrä on ennalta sovittavissa. Jos pyrähdysten määrä päätetään alussa, pyritään silloin myös hahmottelemaan karkeasti työmäärää, jotta pyrähdysten aikana saadaan projekti päätökseen.



Kuvio 1. Havainnollistava kuva Scrum-työskentelystä ja -organisaatiorakenteesta. (Lähde: http://markhoogveld.wordpress.com/tag/scrum-overview/)

Scrum-projekti alkaa ilmenneestä tarpeesta luoda ohjelmisto jonkun ongelman ratkaisemiseksi. Scrum-projektissa tuoteomistaja (engl. *Product Owner*) on se henkilö, joka vastaa kehitettävän tuotteen visioinnista ja kehitystiimin työn arvon maksimoinnista. Usein tuoteomistaja on myös ohjelmistotuotteen tilannut asiakas ja vastaa siitä, että tiimi tekee jokaisessa pyrähdyksessä oikeita asioita.

Tuoteomistaja tietää, mitä ohjelmiston pitää tehdä, ja tämä henkilö huolehtii vaatimusten kommunikoinnista tiimille, muun muassa päivittämällä ja ylläpitämällä tuotteen kehitysjonoa (engl. *Product Backlog*). Tuoteomistaja myös priorisoi kehitysjonon kohteet asiakasarvon mukaan. Tämä tuotteen kehitysjono onkin tuoteomistajan vision ilmentymä. Se on järjestetty lista kaikista niistä ohjelmiston ominaisuuksista, joita saatetaan tarvita. Tuoteomistaja vastaa täysin tämän listan sisällöstä, lisää sinne ominaisuuksia projektin edetessä, järjestää ne ja tarvittaessa poistaa listalta turhiksi osoittautuneita ominaisuuksia.

Tuotteen kehitysjono on siis jatkuvassa muutoksessa, ja tuoteomistaja saa vapaasti tehdä sinne, mitä haluaa. Kun pyrähdys suunnitellaan, valitaan tuotteen kehitysjonosta pyrähdyksessä toteutettavat ominaisuudet. Yksi ominaisuus voi olla vaikka käyttäjän sisäänkirjautuminen. Kun tämä ominaisuus valitaan pyrähdyksessä toteutettavaksi,

voidaan se pilkkoa useammaksi tehtäväksi, kuten tietokannan suunnitteluksi ja implementoinniksi ja graafisen käyttöliittymän suunnitteluksi ja toteutukseksi. Näin tästä yhdestä ominaisuudesta muodostuu neljä erillistä tehtävää, jotka kirjataan pyrähdyksen tehtävälistaan (engl. *Sprint Backlog*) pyrähdyksen suunnittelupalaverissa. Mikäli havaitaan, että ominaisuuden eli tuotteen kehitysjonon kohdan toteuttamiseksi pitääkin tehdä vielä enemmän tehtäviä, voi kehitystiimi vapaasti lisätä, muokata tai poistaa tehtävälistalta asioita. Sen sijaan pyrähdyksen suunnittelupalaverissa sovitut ominaisuudet, eli kehitysjonon kohdat, eivät saa pyrähdyksen aikana vaihtua edes tuoteomistajan toimesta. Pyrähdyksessä tehdään aina loppuun sovitut asiat ja seuraavassa pyrähdyksessä vasta voidaan valita uusia kohtia tai muuttaa koko projektin suuntaa näin halutessa.

Mikäli tuoteomistaja, eli joskus myös asiakas, alkaa häiritsemään tiimiä liikaa, tulee Scrum Master tuoteomistajan ja tiimin väliin ja varmistaa sen, että tiimi saa työskentelyrauhan. Scrum Master myös vastaa siitä, että kaikki noudattavat Scrumin sääntöjä ja että projekti etenee jouhevasti.

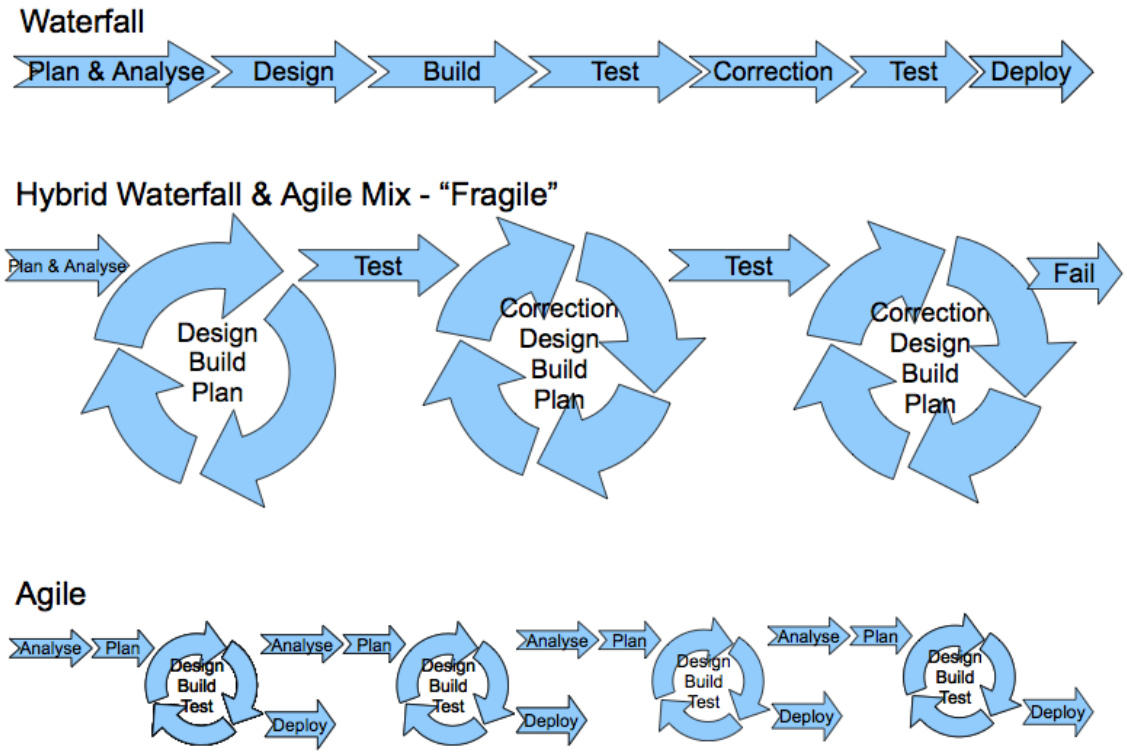
Kun tehtävälista on tyhjä ja kaikki tuotteen kehitysjonosta valitut ominaisuudet on pyrähdyksessä toteutettu, voidaan pyrähdys katsoa päättyneeksi. Sen jälkeen voidaan pitää pyrähdyksen katselmus (engl. *Sprint Review*), jossa tiimi esittelee aikaansaannoksensa tuoteomistajalle. Tuoteomistaja antaa palautetta tehdystä työstä ja ohjaa tiimiä haluamaansa suuntaan. Mikäli on tarpeellista, pidetään myös pyrähdyksen retrospektiivi (engl. *Sprint Retrospective*), jossa puolestaan tarkastellaan varsinaista työskentelyprosessia, ja mahdollisesti tehdään muutoksia seuraavaan pyrähdykseen.

Scrum-projekti päättyy, kun viimeinenkin pyrähdys on ohi ja tuotoksena on valmis ohjelmistotuote, joka joko sisältää tai ei sisällä kaikkia tuoteomistajan vision mukaisia ominaisuuksia. Jos projektin aikana havaittiin lisäkehitystarpeita tai osaa tuoteomistajan vision mukaisista ominaisuuksista ei toteutettu, vaikka ne katsotaan tarpeelliseksi, aloitetaan uusi projekti uusin tavoittein.

2.6 Johtopäätökset ja suhde vesiputousmalliin

Kuten aiemmin mainittiin, alkuperäisen vesiputousmallin idea oli toteuttaa ohjelmistoja jossain määrin inkrementaalisesti. Yleinen käsitys vesiputousmallin toteutuksesta kuitenkin eroaa tästä, ja siksi vesiputousmalli eroaa oleellisesti Scrumista.

Siinä, missä vesiputousmalli pyrkii mahdollisimman tarkasti etukäteen suunnittelemaan koko ohjelmiston, pyritään Scrumissa rakentamaan ohjelmistoa osissa. Näitä osia suunnitellaan ja määritellään vasta, kun niiden toteutus alkaa. Koska Scrumissa valmistuu useita ohjelmistoja tai eri versioita samasta ohjelmistosta, pystytään suuntaa vaihtamaan nopeasti. Siinä vaiheessa, kun Scrumilla tuotetusta ohjelmistosta on jo yksi versio olemassa, ollaan vesiputousmallissa vasta määrittelemässä koko sovellusta.



Kuva 4. Vesiputousmallin ja Scrumin prosessit allekkain esitettynä (Lähde: http://article-stack.com/education/agile-software-development-an-overview-in-5-mins.amty)

Kuten kuvasta 4 nähdään: jos suunta on ollut väärä, menee yleensä korkeintaan yksi pyrähdys hukkaan, toisin kuin vesiputousmallissa, jossa väärä tai huono päätös alkuvaiheessa johtaa pahimmillaan koko projektin epäonnistumiseen tai merkittävään lisätyöskentelyyn.

Toisaalta voidaan miettiä, mahdollistaako Scrum sellaisen kokonaisvaltaisen projektin tavoitteiden ymmärtämisen – kuten vesiputousmalli – sikäli, kun tehtävät suunnitellaan tarkemmin vasta juuri ennen toteutusvaihetta. Mikäli Scrumia käytettäessä ei muodosteta kokonaiskuvaa tuotteesta, voi olla vaikeaa hahmottaa kompleksia kokonaisuutta sen koko toiminnan kannalta. Todellisuudessa kuitenkin Scrumissa luotavat tarinat (engl. *User story*) muodostavat juurikin tilanteeseen liittyvän skenaarion, jonka pitää

toteutua. Nämä ovat aina reaalimaailman esimerkkejä ja kohdistuvat vastaamaan kysymyksiin, kuka ohjelmistoa käyttää ja mitä hän sillä haluaa saavuttaa. Tämän insinöörityön tilaajayritys Toinen veli Oy:ssä määritellään lisäksi aina pyrähdykselle tavoite, joka kirjoitetaan sanamuotoon ylös. Se on käytännössä sanallinen, mielellään yhden virkkeen mittainen seloste siitä, mitä pyrähdyksen jälkeen on valmiina, tarkemmin sanoen, mitä ohjelmistolla voi tehdä ja kenen toimesta. Tällainen tavoitteenasettelu on myös lisätty viralliseen Scrum Guide 2013:sta heinäkuussa 2013 (Lekman 2013). Pyrähdyksen tavoitteenasettelun lisäksi toteutetaan myös suurpiirteisempi tavoite koko projektista.

Yhteenvetona Scrumin ja vesiputousmallin suhteesta voidaan sanoa, että vesiputousmalli antaa varmasti enemmän ennustettavuutta, mutta heikentää reaktiokykyä erinäisissä tilanteissa. Vastaavasti Scrum antaa paljon liikkumavapautta ja virheenkorjauskykyä, mutta tekee ennustamisesta, aina budjetista alkaen, vähintäänkin haastavaa.

3 Lean

Lean on ennen kaikkea ajattelumalli tai johtamisfilosofia, joka ketterän ohjelmistokehityksen julistuksen tavoin pohjautuu tiettyihin arvoihin. Näistä arvoista tärkeimpänä yksittäisenä pidetään sellaista tuotantoa, joka vain ja ainoastaan tuottaa asiakkaalle arvoa. Kaikki muu työ katsotaan jätteeksi ja pyritään poistamaan prosessista kokonaan.

Poppendieck esitti vuonna 2003 julkaisemassaan kirjassaan Lean-ajattelumalliin pohjautuvan Lean Software Developmentin, jossa Lean-ajattelumallin periaatteet sovelletaan ohjelmistokehitystyöhön sopiviksi. Lean Software Development pohjautuu alun perin Toyotan kehittämään Toyota Production Systemiin, ja sen johtoajatus on olla tuotekehitysprosessi, jonka osaksi ohjelmistokehitys mielletään. (Poppendieck & Poppendieck 2003 ja Poppendieck & Poppendieck 2007.)

Aliluvussa 3.2 esitellään lisäksi Helsingin yliopiston tietojenkäsittelytieteen laitoksen alainen organisaatio, joka käyttää Lean-menetelmiä ohjelmistoprojekteissaan. Organisaatio valitsi Kanbanin, tai siitä sovelletun Scrumbanin, ennen kaikkea sen nopean käyttöönoton vuoksi (Fagerholm 2013). Kanban esitellään luvussa 4 ja Scrumban luvussa 5.5.

3.1 Leanin ja Lean Software Developmentin määritelmä

Lean on englanninkielen sana, jonka virallinen käännös suomeksi on hoikka tai virtaviivainen. Lean ei ole prosessimalli tai työkalu, vaan ennen kaikkea ajattelumalli tai johtamisfilosofia. Lean-ajattelua toteuttavia projektinhallinnan prosessimalleja on esimerkiksi Kanban ja **koko Lean-ajattelun ydin on arvoa lisäävä tuotanto**. Kaikkein tärkein ajatus on siis tuottaa arvoa asiakkaalle, ja asiakkaaksi ohjelmistoprojekteissa katsotaan esimerkiksi ohjelmiston käyttäjä.

Jotta arvoa voidaan tuottaa, pyritään poistamaan kaikki turha ja keskittymään vain näihin arvoa tuottaviin asioihin. Alkujaan Toyotan käyttämästä mallista johdettu Lean määrittelee turhaksi kaiken sellaisen toiminnan, jossa käytetään resursseja mihinkään muuhun kuin asioihin, jotka tuottavat arvoa asiakkaalle.

Lean-mallissa haetaan jatkuvasti epäkohtia tuotanto- tai kehitysprosesseissa ja eliminoidaan epäkohdat. Epäkohdat ovat asioita, jotka eivät tuota asiakkaalle lisäarvoa. Ohjelmistokehityksessä työn painoarvo on monistamisen sijaan suunnittelussa tai kehityksessä (Fagerholm 2013). Fagerholmin mukaan ohjelmistotuotantoprosessissa luodaan ja jalostetaan tietoa eikä materiaaleja. Se aiheuttaa ongelmia siinä mielessä, että varsinainen monistaminen, kuten tuotantolinja-ajattelussa, on nopeaa ja mutkatonta. Varsinainen kehitettäväksi jäävä prosessi on nimenomaan suunnitteluun ja kehitykseen liittyvä prosessi. Fagerholm näkee tuotanto- ja kehitysprosessin ongelmana juuri jokaisen projektin tai tuotteet yksilöllisyyden ja kokeekin, että ohjelmistokehityksen maailmassa olisi tärkeämpää keskittyä oppivan organisaation käsitteeseen. Siihen, että miten jo opittua tietoa saadaan jaettua ja näin välttämään uudestaan samojen virheiden tekemistä.

Poppendieck pitää kirjassaan oletusarvoisena sitä, että ohjelmistot pitäisi aina rakentaa tuotteiksi (Poppendieck & Poppendieck 2007). Asiakkaat eivät halua ostaa ohjelmistoja vaan tuotteita. He haluavat ratkaisuja ongelmiinsa tai lisäarvoa jostain tuotteesta. Ohjelmistoja ei osteta ohjelmiston vuoksi, vaan johonkin liiketoiminnan käyttöön. Siksi ohjelmistojen kehityksessä pitäisi ensisijaisesti nojata tuotekehitysmalleihin.

Lean Software Developmentissa (LSD) ajatus ohjelmistojen kehityksestä siirretään tuotteiden kehittämiseen, varsinainen ohjelmistokehitys ajatellaan vain osana tuotekehitysprosessia. Sen takia hyvä tuotekehitysprosessi toimii yhtä lailla hyvän ohjelmisto-

kehitysprosessin mallina. Näistä Lean-mallin mukaiseksi hyväksi tuotekehitysprosessiksi on otettu Toyota Product Development System.

Lähtökohtainen ajatus LSD:ssa on se, että ohjelmiston pitää olla helposti muutettavissa. Ohjelmistoja rakennetaan ratkaisemaan ongelmia, jotka ovat fyysisesti hankalasti muutettavissa, joten ohjelmiston pitää olla aina sillä tasolla, että se pysyy helposti muutettavana.

3.2 Esimerkkiorganisaatio: Software Factory

Software Factory (SF) on Helsingin yliopiston tietojenkäsittelytieteen laitoksen alainen laboratorio, jossa opiskelijat kehittävät asiakkaiden tilauksesta ohjelmistoja Kanban/Scrumban-menetelmää käyttäen. Ohjelmistot kehitetään reaalimaailman tarpeisiin, ja projektien tuotoksina syntyy tyypillisesti prototyyppejä johonkin uuteen liiketoiminta-alueeseen, mutta joskus myös tuotantokäyttöön soveltuvia ohjelmistoja.

Tämän insinöörityön kolmannessa kappaleessa viitataan usein SF:n operatiivisesta toiminnasta vastaavan Fabian Fagerholmin kanssa käytyyn keskusteluun heidän käyttämistä menetelmistä ja hyväksi havaituista käytännöistä.

Mainitsemisen arvoista on, että asiakkaat, jotka tilaavat ohjelmistoprojekteja SF:ltä, eivät maksa tilauksestaan mitään, joten jotkin ilmiöt eroavat varsinaisesta liiketoimintaympäristöstä. Esimerkiksi SF:n asiakkaiden ominaisuuksien priorisointi voi olla löyhempää siksi, ettei sata ominaisuutta maksa yhtä enempää. On kuitenkin huomioitava, että SF:n projektit kestävät vain seitsemän viikkoa, joten ajan rajallisuuden kautta asiakkaille syntyy halu toteuttaa aina tärkeimmät ominaisuudet annetussa aikaviipaleessa.

3.3 Leanin yleiset periaatteet

Poppendieck esitti vuonna 2007 kirjoittamassaan kirjassa seitsemän Leanin yleistä periaatetta ja konkretisoi niiden merkityksen ohjelmistokehitystyössä. Periaatteita ovat

- jätteen eliminointi, jossa pyritään poistamaan kaikki tuottamaton työ
- alusta asti laadukkaiden ohjelmistojen rakentaminen, jossa valmistuvat ohjelmiston osat ovat toimivia ja testattuja

- tiedon luominen, jonka kantava ajatus on ketteristä ohjelmistokehitysmenetelmistä tuttu – prosessin aikana syntyvän informaation hyödyntäminen
- sitoutumisen viivyttäminen, jonka mukaan sitovat päätökset pitäisi tehdä niin myöhään kuin suinkin mahdollista
- nopeasti toimittaminen, minkä ansiosta resurssit vapautuvat nopeasti ja työajan tehokkuus paranee
- ihmisten kunnioittaminen, jonka on oltava keskiössä, kun tavoitteena on luoda laadukasta ohjelmistoa
- kokonaisuuden optimointi, jossa keskitytään osaoptimoinnin sijaan koko arvovirtaan ja sen parhaaseen mahdolliseen suoritukseen.

Aliluvuissa käsitellään näitä periaatteita tarkemmin ja kerrotaan, mitä ne käytännön ohjelmistokehitystyössä kukin merkitsevät.

3.3.1 Eliminoi jäte

Taiichi Ohno sanoi Toyota Production Systemin (TPS) olevan johtamisjärjestelmä, joka absoluuttisesti eliminoi kaiken jätteen. Ajatus TPS:ssä on se, että tarkastellaan aikajanaa, joka alkaa asiakkaan tekemästä tilauksesta siihen hetkeen, kun asiakasta laskutetaan valmiista työstä. Tätä aikajanaa pyritään jatkuvasti lyhentämään poistamalla jätettä, joka ei tuota arvoa.

LSD:ssa ajatus on sama, mutta aikajanan alku- ja loppuhetkeä voi muokata. Aikajana alkaa, kun tilaus tulee sisään, ja päättyy siihen hetkeen, kun se on toimitettu. LSD keskittyy lyhentämään sitä aikajanaa poistamalla kaiken arvoa tuottamattoman jätteen.

Fagerholm näkee SF:n toiminnassa epäkohtina turhien ominaisuuksien tuottamisen, koska jokainen koodirivi on ohjelmistolle rasite sen koko elinkaaren ajaksi. Esimerkkinä ovat kokoukset, jotka pidetään vain siksi, että ne pitäisi pitää, koska ne eivät tuota mitään muuta kuin illuusion siitä, että jokin asia on nyt kokouksessa hoidettu. Myös ylitöiden tekeminen rasittaa tiimiä.

Mikä lasketaan jätteeksi? Tuotantoympäristöissä sellaiseksi katsotaan varastot. Varastoja pitää hoitaa, siirtää, hakea ja järjestellä. Se ei pelkästään lisää kustannuksia vaan myös kompleksisuutta – joka on suuri kustannuskertoja. Varasto hukkuu, vanhenee ja sitoo rahaa.

Ohjelmistokehityksessä jätettä on osittain tehty työ. Varaston tavoin se hukkuu, vanhenee ja sitoo rahaa. Ylitse muiden asioiden kaikkein suurin jäte tulee ylimääräisistä toiminnallisuuksista. Vain noin 20 % ominaisuuksista ja toiminnallisuuksista räätälöidyssä ohjelmistossa on sellaisia, joita käytetään säännöllisesti. Noin 67 % toiminnallisuuksista on sellaisia, joita käytetään vain harvoin. (Poppendieck & Poppendieck 2007.)

Vähän käytetyiksi tai ylimääräisiksi toiminnoiksi ei lueta mukaan esimerkiksi ohjelmiston turvallisuutta parantavia ominaisuuksia vaan ominaisuuksia, joita ei alun alkaenkaan ole tarvittu. Kun käyttämättä jääviä toiminnallisuuksia toteutetaan, ne paitsi vievät resursseja toteutusvaiheessa, mutta ennen kaikkea lisäävät kompleksisuutta ohjelmakoodiin, joka taas kasvattaa joka kerta ohjelmiston ylläpidon kustannuksia.

3.3.2 Rakenna laatua

Ajatus LSD:ssä on rakentaa laadukkaita ohjelmistoja ensimmäisistä koodiriveistä lähtien. Fokus siirtyy virheiden kirjaamisesta ja huomaamisesta virheiden esiintymisen poistamiseen.

Jos koodi testataan vasta sen jälkeen, kun se on kirjoitettu, todennäköisesti siinä esiintyy virheitä. Mikäli virheet kirjataan ylös, syntyy lista osittain tehdystä työstä, jonka katsotaan olevan jätettä. Poppendieck väittääkin menestyvien LSD:tä käyttävien organisaatioiden käyttävän myös TDD:tä hyvin laajasti, jotta ohjelmisto on testattua silloin, kun sen katsotaan olevan valmis.

3.3.3 Luo tietoa

Vesiputousmallin ongelma on se, että siinä ajatellaan määrittelyvaiheessa kaiken tiedon olevan jo saatavilla. Todellisuudessa toteutusvaiheessa esiintyy asioita, jotka muuttavat käsitystä luotavasta ohjelmistosta tai tuotteesta. Ajatus LSD:ssä, kuten muissakin ketterissä menetelmissä, on hyödyntää tätä kehityksen aikana syntynyttä tietoa ja päättää toteutuksen suunta tämän tiedon ollessa saatavilla.

LSD:n yksi johtoajatuksista on se, että koko prosessin pitää kannustaa oppimaan aiemmin tehdystä työstä, mutta myös parantamaan kehitysprosessia. Lean-organisaatio tietää, että sen pitää aina parantaa prosessejaan, koska monimutkaisissa järjestelmissä on ja tulee aina olemaan ongelmia. Jokaisen virheen tulee synnyttää

etsintäprosessi, jossa virheen alkuperäinen syy jäljitetään, ja sen jälkeen muutetaan prosessia niin, ettei sama virhe pääse toistumaan.

3.3.4 Viivytä sitoutumista

Ajatus viivyttämisestä on uusi mutta tärkeä. LSD:ssä sellaiset päätökset, jotka ovat lopullisia, pitäisi tehdä niin myöhään kuin vain mahdollista. Se ei tarkoita sitä, että kaikki päätökset pitäisi tehdä viimeisellä hetkellä. Päätökset, jotka eivät ole peruuttamattomia, pitäisi tehdä nopeasti, ja tarvittaessa niitä pitää muuttaa.

Oleellista on se, että peruuttamattomat päätökset kannattaa tehdä viimeisellä hetkellä siksi, että silloin on kaikkein suurin tieto saatavilla. Tällaisia päätöksiä voivat olla tarkat rajanvedot arkkitehtuurista. Kun saatavilla on kaikki opittu tieto tähän asti tehdystä työstä, on päätös todennäköisesti paljon parempi kuin sellainen, joka tehtäisiin heti kehitysprojektin alkaessa.

3.3.5 Toimita nopeasti

Mitä nopeammin ohjelmiston pystyy toimittamaan, sitä nopeammin resurssit vapautuvat muuhun käyttöön ja sitä nopeammin saadaan rahat asiakkailta. Nopeasti toimittaminen on myös asiakkaiden kannalta kriittistä, sillä pitkissä projekteissa on riskinä, että toimintaympäristö, johon ohjelmisto tulee, ehtii muuttumaan ennen kuin järjestelmää toimitetaan. Silloin ohjelmisto on jo toimitusvaiheessa vanhentunut.

Nopeasti toimittaminen jatkuvalla tahdilla ei kuitenkaan ole mahdollista ilman loistavaa laatua.

3.3.6 Kunnioita ihmisiä

Ihmiset, jotka tuntevat olevansa kunnioitettuja, tuottavat kerta toisensa jälkeen loistavia tuloksia. Ihmisten kunnioitus LSD:ssä tarkoittaa ennen kaikkea sitä, että sen sijaan, että ihmisille kerrotaan, mitä tehdä, kehitetään organisaatio, jossa ihmiset käyttävät itse arviointikykyään ja arvioivat itse, mitä on milloinkin tehtävä.

3.3.7 Optimoi kokonaisuutta

Lean-organisaatio optimoi koko arvovirtaa siitä hetkestä, kun se saa asiakkaalta tilauksen aina siihen hetkeen, kun asiakkaan tarve on tyydytetty. Jos organisaatio keskittyy optimoimaan jotakin yksittäistä toimintoa tai prosessin osaa, koko arvovirta kärsii.

4 Kanban

Kanban on Lean-ajattelua toteuttava prosessimalli. Kanban on prosessimallina adaptiivinen, joka tarkoittaa sitä, että se ei itsessään määrittele paljoa sitä, millainen prosessin pitäisi olla.

Esimerkiksi Scrum määrittelee tarkat roolit tiimille ja asettaa jopa suurpiirteiset vaatimukset tiimiä ympäröivälle organisaatiolle, kun taas Kanbanilla on vain kolme vaatimusta: visualisoidaan työn vuo (engl. *flow*), rajoitetaan samaan aikaan tehtävien töiden määrä (engl. *Work-in-Progress, WIP*) johonkin kiinteään määrään ja mitataan läpimenoaikaa.

Kolmen vaatimuksen ansiosta Kanban on teoriatasolla hyvin löyhästi määritelty ja sen voi ottaa helposti käyttöön, koska sen opettelu ei vaadi paljoa aikaa. Jos organisaatio haluaa lisäsääntöjä tai tarvitsee sovittuja asioita, kukaan ei kiellä lisäämästä sellaisia. Oleellinen ero on siinä, että Kanban ei itsessään määrittele edellä mainittuja kolmea sääntöä enempää.

4.1 Kanbanin historia ja vaiheet

Lean ja Kanban esiteltiin Japanin tuotannollisessa teollisuudessa jo 1950-luvulla, ja Kanban onkin japaninkielen sana ja tarkoittaa kaikessa yksinkertaisuudessaan merkkitaulua (eng. *signboard*). Sellainen se pääpiirteittäin onkin. Ohjelmistokehityksessä Kanban sai alkunsa vuonna 2004, kun David J. Anderson avusti Microsoftilla työskennellyttä pientä IT-tiimiä, joka suoriutui heikosti. Tiimillä oli tuotteen kehitysjonossa useita kymmeniä toiminnallisuuksia, ja heidän keskimääräinen läpimenoaikansa (engl. *Lead Time*) oli miltei puoli vuotta. Anderson mietti vaihtoehtoja ja päätti kokeilla Kanbania tässä tiimissä. Hän lupasi, että tiimi toimitti jatkossa yhden toiminnallisuuden vain 25 päivässä, eli vain kuudesosassa siitä ajasta, mikä tiimiltä aiemmin kesti toimittaa.

Niinpä Anderson Lean-mallin mukaisesti pyrki poistamaan kaiken arvoa tuottamattoman työn, kuten yksittäisten kohteiden työmääräarvioinnin ja sen sijaan keskittyi ominaisuuksien priorisointiin ja läpimenoajan mittaamiseen. He pysyivät 25 päivän läpimenoaikalupauksessaan 90-prosenttisesti. (Kanban: successful evolutionary change for your technology business – David J. Anderson 2013.)

Joskus Kanban liitetään virheellisesti Toyota Production Systemiin (*TPS*), koska kun Kanbanin sääntöjä tulkittiin, tulkittiin niitä toisaalta tuotannon säännöiksi, mutta toisaalta myös Kanbanin säännöiksi. Esimerkiksi Wikipedian Kanban-artikkelissa viitataan Toyotan käyttämiin Kanban-sääntöihin, joita on kuusi kappaletta. Kuten edellisessä luvussa mainittiin, Toyota on itse päättänyt luoda sääntöjä, joita Kanban ei kuitenkaan *edellytä*.

Shingo esimerkiksi raportoi TPS:än olevan käytännössä 80-prosenttisesti jätteen eliminointia, 15 % tuotantojärjestelmää ja 5 % Kanbania.

Oulun yliopistossa tehdyn kirjallisuuskatsauksen mukaan vasta vuoden 2007 jälkeen on tehty tutkimuksia Kanbanista ohjelmistokehitystyössä. Vuodesta 2008 tutkimusten määrä on ollut selvässä kasvussa, vaikka vuonna 2011 Kanbania ohjelmistokehityksessä koskevia tutkimuksia oli julkaistu vain kymmenen. Kirjallisuuskatsauksesta johtopäätöksenä voidaan tulkita, että Kanbanin käyttö tai ainakin kiinnostus Kanbanin käyttöön ohjelmistokehityksessä on kasvussa.

Tätä insinöörityötä tehdessä Kanban ja Lean herättävät allekirjoittaneessa vahvaa mielenkiintoa, mutta mitä pidemmälle työ etenee, sitä useammin tuntuu nousevan henkilökohtainen ajatukseni siitä, ettei Kanbankaan ole hopealuoti (engl. *Silver Bullet*), mutta varmasti tietyntyyppiseen ohjelmistokehitykseen toimiva malli ja Lean-ajatusideologia varmasti muita ohjelmistokehitysprosessimalleja oivasti täydentävä.

4.2 Kanban-prosessimalli

Kanban-taulussa visualisoidaan työnkulku. Sen voi tehdä fyysiselle taululle tai jollain ohjelmistolla. Oleellista on se, että yhdellä silmäyksellä nähdään projektin tai prosessin kulloinenkin tilanne. Taulu jaetaan vaiheisiin, joita on yksinkertaisimmillaan kolme, kuten esimerkiksi tehtävälista, työn alla ja valmistunut. Tarpeen vaatiessa näitä vaiheita, eli Kanban-taulun sarakkeita, voi olla useita. Kirjallisuudesta ei löydy kiinteästi asetet-

tua enimmäismäärää sarakkeiden määrälle, mutta sen pitäisi aina olla käyttötarkoitukseltaan looginen ja tietynlaisen suuruusrajoitteen asettaa se, että koko taulu pitää pystyä näkemään kerralla. Oikea sarakkeiden määrä taas riippuu siitä prosessista, jota sillä kuvataan. Jos Kanbania käytetään ohjelmistokehityksessä, jossa varsinaisen toteutuksen jälkeen on vielä käyttöönotto, niin sille voi olla oma sarakkeensa.

Kuva 5. Esimerkki Kanban-taulusta. (Lähde: http://blog.vincity.fi/2013/06/)

Kanban-projektin alkaessa määritellään ensin työmääräraja (engl. *Work-in-Progress limit*), joka pitää suhteuttaa tiimin jäsenten lukumäärään ja tiimin yleiseen suorituskykyyn. Minkäänlaista laskukaavaa tähän ei ole olemassa, vaan se täytyy määritellä aina tapauskohtaisesti. Jos esimerkiksi käytetään pariohjelmointia, silloin on luonnollista, että raja on puolet pienempi kuin sellaisessa tilanteessa, jossa sitä ei käytetä.

Työmääräraja asettaa maksimimäärän yhdessä sarakkeessa samanaikaisesti olevia tehtäviä. Jos esimerkiksi työn alla -kohdassa on rajana kolme, niin kolmea tehtävää enempää siellä ei saa samaan aikaan olla. Jonkun tehtävän on siis valmistuttava, ja se on siirrettävä seuraavalle sarakkeelle ennen kuin uutta tehtävää voidaan sarakkeeseen tuoda.

Yleinen nyrkkisääntö työmäärärajan asettamisessa on, että valitaan ensin kohtalaisen pieni raja. Kuten Kanbaniin kuuluu, prosessia jatkuvasti tarkkaillaan ja kehitetään ja jos havaitaan, että raja vaatii muutoksia, niin kehityksen seurauksena rajaa muutetaan.

Kanbanissa keskitytään työn vuohon (engl. *Flow*), joka viittaa ohjelmistotuotteen koko arvovirtaan, eli siihen prosessiin, joka alkaa idean, tai jonkin ongelman ratkaisun syntymisestä, kulkee läpi toteutuksen, testauksen ja käyttöönoton. Tämä vuo visualisoidaan Kanban-taulun avulla niin, että siitä näkee heti, mitä milloinkin tapahtuu.

Tehtävät siirtyvät vaiheesta toiseen, ja yksi vaihe tehdään aina valmiiksi ennen tehtävän siirtämistä seuraavaan vaiheeseen. Kaikkein tärkein asia vuota hallitessa on se, että pyritään eroon jätteestä (engl. *waste*, japaniksi *Muda*). Jätteen tarkempi määritelmä ohjelmistotuotannossa on selitetty aiemmin luvussa 3.3.1, mutta lyhyesti se tarkoittaa mitä tahansa sellaista aktiviteettia, jonka seurauksena ei synny arvoa asiakkaalle.

Kun tehtävät liikkuvat Kanban-taululla, niitä ei koskaan työnnetä seuraaviin vaiheisiin esimerkiksi projektipäällikön toimesta, vaan tiimi aina valitsee vasemmalla olevasta sarakkeesta seuraavan työn alle otettavan tehtävän. Tätä kutsutaan Just-in-Time-mekanismiksi, jossa tiimi aina pyytää (engl. *pull*) valitsemansa tai tarvitsemansa tehtävän ja toteuttaa sen. Näin ollen koodia ei synny varastoon implementointia odottamaan, koska varastot tai käyttämättä oleva koodi katsottaisiin jätteeksi. Joskus Just-in-Time-mekanismia kutsutaan Suomessa myös nimellä Juuri Oikeaan Tarpeeseen ja siitä voidaan käyttää lyhennystä JOT (Logistiikan Maailma 2013).

4.3 Kanban nykyhetkessä

Kanbanin suosio ohjelmistokehityksessä kasvaa, kuten Oulun yliopiston kirjallisuuskatsaus osoittaa, mutta sen käyttöaste on edelleen verraten vähäistä (Ahmad ym. 2013). Kanban itsessään on taulu, jolla työn kulku visualisoidaan, mutta siihen liitetään aina Lean-ajattelufilosofia. Ongelmia muodostuu, jos organisaatio implementoi taulun käsittämättä taustalla olevia ajattelumalleja.

Oulun yliopiston kirjallisuuskatsauksen perusteella Kanbanin suurin haaste on se, ettei Kanban yksin riitä, vaan tarvitsee tuekseen muita ketterien ohjelmistokehitysmenetelmien käytäntöjä. Lisäksi kirjallisuuskatsauksessa läpikäydyissä tutkimuksissa toiseksi suurimmaksi haasteeksi koitui organisaation kulttuuri- ja filosofiamuutosten läpiviemi-

nen. Tämä on luonnollinen haaste minkä tahansa uuden prosessin sisäänajovaiheessa. Kolmanneksi suurimpana haasteena koettiin erityistaitojen tai koulutuksen puute ja niiden seurauksena se, ettei Kanbanin ydinperiaatteita aina ymmärretä.

5 Ketterien menetelmien ongelmat/haasteet

Ketterät ohjelmistokehitysmenetelmät soveltuvat parhaiten juuri sellaisiin projekteihin, joissa luodaan jotain täysin uutta. Ne mahdollistavat prosessinaikaisen oppimisen hyödyntämisen.

Toisaalta prosessinaikainen muutoskyky luo haasteita projektien hinnoitteluun ja ketterä ohjelmistokehitys voikin olla vaikeasti perusteltavissa asiakkaalle, ellei asiakkaan kanssa ole synnytetty luottamussuhdetta. Ketterien menetelmien käyttäminen ei myöskään hyödytä, jos niiden taustalla olevia arvoja ei ymmärretä tai asiakasta ei saada ymmärtämään mallien toimintaperiaatteita ja syitä.

Luvussa 6 esitettyjen tutkimustulosten perusteella Kanban tarvitsee tuekseen muitakin menetelmiä ja joissain organisaatioissa onkin päädytty Scrumin ja Kanbanin yhdistävään hybridimalliin. Scrumin ja Kanbanin hybridimallia kutsutaan Scrumbaniksi, jossa pyritään yhdistämään molempien mallien parhaat puolet.

5.1 Ketterien menetelmien soveltuvuus

Steve Jobs on sanonut, että ihmiset eivät yleensä tiedä, mitä haluavat, kunnes näytät sen niille (Bloomberg 1998). Ohjelmistokehityksessä suurin haaste onkin tietää, mihin ongelmaan asiakas haluaa ratkaisun. Vaikka asiakas osaisi kuvailla ongelman ja jopa esittää valmiin ratkaisuehdotuksen, se ei silti toteutuessaan automaattisesti hyödytä asiakasta. Sen sijaan ohjelmistokehityksen palveluvalikoiman tulee pitää sisällään myös pelkän toteutettavan ohjelmiston lisäksi varsinaisen ongelman ratkaiseminen.

Jo aiemmin Jobs ymmärsi, että vaikka asiakkaalle esitettäisiin suoria kysymyksiä, niin toimimalla asiakkaan antamien vastausten mukaan ohjelmisto mitä todennäköisimmin ei olekaan halutunlainen (Birlingham & Gendron 1989). Tähän ongelmaan ketterät, inkrementaalisesti ja iteratiivisesti toteutettavat kehitysmenetelmät tarjoavat loistavan ulospääsyn. Siinä ohjelmisto rakennetaan valmiiksi, jonka jälkeen sillä voidaan toteut-

taa vain rajattu määrä asioita, mutta se näytetään asiakkaalle. Tässä vaiheessa asiakas pääsee näkemään ja kokemaan ohjelmiston ja antamaan palautetta siitä.

Vesiputousmalli on järkevin vaihtoehto silloin, kun kaikki mahdolliset taustatiedot ovat selvillä. Taustatietoja ovat ainakin

- tarkka käsitys siitä, mitä ohjelmiston pitää tehdä
- selkeä kuvaus siitä, miten ohjelmiston pitää toimia
- käsitys siitä, minkälaisessa ympäristössä ohjelmistoa vain ja ainoastaan käytetään
- tieto siitä, ketkä tarkalleen ottaen ohjelmistoa käyttävät ja mitkä heidän taustansa ovat.

Edellä mainitut monet esivaatimukset toteutuvat tuskin koskaan isoissa projekteissa, mutta voivat hyvin toteutua pienemmissä projekteissa. Jos jokin yksinkertaisempi asia halutaan esimerkiksi automatisoida ja esivaatimustiedot tiedetään, niin silloin projektin jakaminen pyrähdyksiin ei todennäköisesti ole kaikkein tehokkainta.

Yleensä ohjelmistoprojektit aloitetaan jonkin kompleksin asian ratkaisemiseksi tai yksinkertaistamiseksi, jolloin esivaatimustiedot eivät ole todennäköisesti alkuvaiheessa tiedossa ja ne voivat herkästi muuttua. Silloin ketterä ohjelmistokehitys toimii parhaiten.

5.2 Ketterien ohjelmistoprojektien hinnoittelu

Eräs iso ongelma kaikissa ketterissä ohjelmistoprojekteissa ja etenkin Lean-menetelmällä tuotetuissa projekteissa on niiden hinnoittelu ja myyminen asiakkaalle (Peurala 2013). Mikäli asiakas haluaa kiinteähintaisen projektina tuotetun ohjelmistoratkaisun, niin lähtökohtaisesti sellaisen toimittaminen ei pitäisi sopimusteknisesti olla mahdollista ketterälle projektille (Jaakonaho 2009). Sopimus kiinteästä summasta pitää sisällään myös määrittelyn ohjelmistosta. Tilanteesta riippuen määrittely voi sisältää hyvinkin tarkat raamit ulkoasusta, toiminnoista tai muista vastaavista seikoista (Jaakonaho 2009).

Jos toimittaja pitää kiinni tarkasti määritellystä sopimuksesta, voidaan toki työ jakaa pyrähdyksiin ja asettaa välitavoitteita, mutta käytännössä suunnan vaihtaminen ei ole mahdollista – ellei sille ole sopimuksessa keinoja. Sellaisessa tilanteessa asiakkaan

antama palaute voi olla täysin hyödyntämättömissä. Myös kiinteään hintaan sidottuna se estää aidosti paneutumasta asiakkaan rooliin ja ratkaisemasta asiakkaan todellisia liiketoiminnallisia ongelmia.

Asiakkaan kannalta tilanne on myös hankala, koska asiakas haluaisi toki tietää, mitä hän on ostamassa. Fagerholmin mukaan Software Factory pystyy antamaan seuraavan lupauksen (Fagerholm 2013):

> ”Saatte jotain mistä on hyötyä aikataulun loppuun mennessä, mutta me emme tiedä vielä mitä se on, mutta teemme parhaamme.”

Sitaatti onkin se, mitä ketterästi toimiva toimittaja voi aidosti luvata. Näillä lupauksilla tehtyä kauppaa ei ole järin helposti ostajan muotoiltavissa johdolle, ainakaan jos toimittaja ei aiemmin ole synnyttänyt syvää luottamussuhdetta asiakkaaseen.

Lisäksi Leanissa ei lähtökohtaisesti aseteta aikarajoja (Petersen & Wohlin 2009), joten tehtävä valmistuu, kun valmistuu. Näin ollen asiakkaalle ei voida Leanissa kertoa, että tämä ja tämä toiminto on valmis tähän päivämäärään mennessä, toisin kuin Scrumissa.

Toisaalta taas Leanissa seurataan läpimenoaikaa ja siitä voidaan ammentaa asiakkaallekin tärkeitä seikkoja, kuten ”yleensä tämänkaltainen toiminto toteutetaan kahdessa päivässä”. Tämän ansiosta projektien lukumäärän kasvaessa pystytään antamaan suuntaa-antavia arvioita asiakkaalle kustannustasosta.

Kiinteä hinnoittelu sopii silti erittäin huonosti Leaniin (Jaakonaho 2009), ja siksi moni yritys laskuttaakin tehtyjen työtuntien mukaan, kuten Fraktio Oy, jonka toimitusjohtaja kertoo mahdollisten yhteistyökeskustelujen alkavan tuntihintojensa esittelystä ja jos se ei muodostu ongelmaksi, niin varsinaiset neuvottelut voivat käynnistyä (Peurala 2013).

5.3 Teorian ja käytännön toteutuserot

Usein teoria ja käytäntö eroavat toisistaan, eikä aina ole edes järkevää toteuttaa teoriaa käytännössä pilkuntarkasti. Kuten aiemmin mainittiin, Lean on ensisijaisesti ajattelumalli ja filosofia, joten se ei itsessään määrittele käytännön toteutusta.

Samalla tavalla kuin ketterän ohjelmistokehityksen julistus, myös Lean nostaa esiin asioita filosofisella tasolla, mutta niistä yksi vapaasti muotoiltuna on ylitse muiden: Tee

vain niitä asioita, jotka hyödyttävät asiakkaita ja pyri jatkuvasti tehostamaan näiden asioiden tekemistä.

Kolme keinoa, joilla tähän käytännössä päästään ovat:

1 Opettele ymmärtämään aidosti asiakkaasi liiketoimintaa ja ole valmis muuttamaan tai ehdottamaan asiakkaalle tämän prosessien muuttamista. Jos ohjelmistolla halutaan ratkaista jokin ongelma, niin selvitä, miksi ongelma on ylipäätään olemassa ja voiko sen poistaa muilla keinoilla.

2 Pidä jatkuvasti yhteyttä asiakkaaseen ja vaadi asiakas osallistumaan tuotteen kehitystyöhön aktiivisesti.

3 Pyri aina kyseenalaistamaan tekemiänne päätöksiä ja keskity poistamaan tai automatisoimaan sellaiset työt, jotka voi automatisoida.

5.4 ScrumBut

ScrumBut on leikkisä termi sellaiselle toiminnalle, jossa Scrumia ei käytetä ohjekirjan mukaisilla tavoilla. Sana tulee englanninkielisestä lauseesta ”We use Scrum BUT…”, joka tarkoittaa sitä, että noudatetaan Scrumin joitain prinsiippejä, mutta ei kaikkia (Rinko-Gay 2013).

Ongelmaksi tämä muodostuu, jos organisaatio pyrkii käyttämään Scrumia, mutta ei ole valmis hyväksymään ketterää ohjelmistokehitysmallia ja sen tuomia lainalaisuuksia (Rinko-Gay 2013). Luonnollisesti termejä voisi keksiä yhtälailla Kanbanista tai mistä tahansa muusta mallista - oleellista näissä kaikissa on aina se, että prosessista otetaan käyttöön vain osia ymmärtämättä koko kontekstia.

Mahdollisia tunnistamistapoja sille, onko organisaatiosi ScrumBut löytyy verkosta (Rinko-Gay 2013), mutta niitä ei käydä tässä työssä tarkemmin läpi.

5.5 Scrumban

Scrumban on menetelmä, jossa käytetään sekä Scrumia että Kanbania. Se voikin olla helppo tapa organisaatiolle siirtyä Kanbaniin tai ottaa käyttöön Kanbanista hyviä puolia ja Scrumista taas toisia hyviä puolia (Pahuja 2012). Yksi Scrumbania käyttävistä orga-

nisaatioista on aiemmin esitelty SF, jonka vetäjä Fagerholm näkee Scrumin ja Lean-ajattelun sopivan hyvin yhteen (Fagerholm 2013).

Mikäli Scrum ja Kanban yhdistetään, tulee vastaan joitakin haasteita. Kuten se, että käytetäänkö Scrumin pyrähdyksen tehtävälistaa, joka pysyy samana pyrähdyksen ajan ja tyhjennetään sen jälkeen, vai annetaanko asiakkaan jatkuvasti tuoda lisää tehtäviä pyrähdykseen, jolloin siitä muodostuu Kanban-taulun yksi sarake (Fagerholm 2013).

Scrumban-menetelmässä pitää tarkkaan sopia, mitä osia otetaan Scrumista ja mitä Kanbanista. Luonnollista olisi sopia vaikka Scrumin pyrähdykset tavoitteineen, mutta sen jälkeen toteuttaa pyrähdys Kanbania käyttäen. Silloin voi myös olla loogista käyttää ainakin joitakin Scrumin roolituksia hyväkseen (Fagerholm 2013).

Scrumbania käytettäessä on otettava huomioon, että se edellyttää vahvaa tuntemusta molemmista menetelmistä, niin Scrumista kuin Kanbanista. Ensinnäkään nykyisellään ei ole saatavilla käsikirjaa Scrumbanin käyttöönottoon, joten valmiita vastauksia tai tarkkoja määritelmiä siitä, miten se pitäisi tehdä, ei ole saatavilla. Toisekseen Scrumbanin käyttö rikkoo aina ainakin Scrumin sääntöjä, mutta joskus toki myös Kanbaninkin.

Haasteista huolimatta sopeuttamalla nämä kaksi toisilleen sopivaa menetelmää yhteen voi olla järkevää. Tutkimustuloksista (Murphy, Bird ym. 2013) käy ilmi, että Kanban tarvitsee tuekseen muitakin metodologioita eikä se yksinään riitä. Mikäli Scrumia jo käyttävä ohjelmistokehitysorganisaatio pyrkii ottamaan Kanbanin käyttöön, voi olla järkevää siirtyä ensin Scrumbaniin ja siitä mahdollisesti eteenpäin Kanbaniin.

Tukholmalaisen yliopiston tapaustutkimuksessa (Nikitina, Kajko-Mattsson, Stråle 2012) tällainen siirtymä toteutettiin ja sen johtopäätöksenä todettiin, että organisaatio voi parantaa prosessia ja saavuttaa kestäviä tuloksia, kun: 1) muodostetaan mekanismit prosessin jatkuvaan parantamiseen ja 2) kun organisaatiossa työskentelee hyvin koulutettu ja sitoutunut henkilökunta. Johtopäätökset todettiin riippumatta siitä, otettiinko käyttöön uusia metodeja vai ei. Todennäköistä silti on, että Kanbanin yksi perusprinsiipeistä, jatkuva prosessin parantaminen, ajoi organisaation hakemaan jatkuvan prosessin parantamisen mallia.

6 Ketterien menetelmien hyödyt

Ketterät ohjelmistokehitysmenetelmät yleistyivät Yhdysvalloissa vuonna 2007 (Fagerholm 2013), joten menetelmän mahdollisista hyödyistä pitäisi olla tutkimuksellista näyttöä, joskin pelkkää hyötyä on vaikea yksiselitteisesti mitata tai edes määritellä.

Microsoft teki laajan viisiosaisen kyselytutkimuksen 1969 henkilölle vuosien 2006 ja 2012 välillä (Murphy ym. 2013). Tarkoitus oli selvittää, miten ketterät ohjelmistokehitysmallit otetaan vastaan ja miten niitä hyödynnetään.

Tutkimuksesta käy ilmi, että henkilöt, jotka käyttävät ketterän ohjelmistokehitystä, ovat siihen erittäin tyytyväisiä (Murphy ym. 2013). Toisaalta tutkimus osoitti myös, että eniten käyttäjiä oli yleensä sellaisilla ohjelmistokehitysmalleilla, joille on hyviä työkaluja käytössä, vaikka ketterän ohjelmistokehityksen julistuksessa työkaluille annetaan vähemmän arvoa kuin yksilöille ja kanssakäymiselle (Beck ym. 2001).

Hyvistä työkaluista huolimatta Microsoftin tutkimuksessa todettiin vastaajien kokevan ketterät ohjelmistokehitysmenetelmät ongelmallisina isojen ohjelmistoprojektien kohdalla (Murphy ym. 2013).

Oslon yliopistossa tehtiin empiirinen tutkimus, jossa tutkittiin erään ohjelmistokehitysyrityksen siirtymistä Scrumista Kanbaniin. Tutkimus osoitti Kanbanista olevan merkittäviä hyötyjä.

Tutkimuksen mukaan Scrumista Kanbaniin siirtyneen organisaation bugien korjaamisen läpimenoaika laski 12 päivän keskiarvosta viiteen, projektin kehitysjonokohteiden läpimenoaika 14 päivästä seitsemään. Keskimäärin siis läpimenoaika lyheni 50% Kanbania käytettäessä aiempaan Scrumiin verrattuna.

Kun mitattiin ohjelmakoodirivejä lisäysten, muokkausten tai poistojen osalta, havaittiin, että virheiden korjaamiseen tuotettiin 6 % enemmän koodirivimuutoksia. Toisaalta taas kehitysjonokohteissa tuotettujen koodirivimuutosten määrä väheni 12 % Kanbanissa.

Tutkimuksessa bugit painotettiin niiden vakavuuden perusteella neljään eri luokkaan;

- julkaisun estäviin bugeihin (8)

- kriittisiin bugeihin (4)
- normaaleihin bugeihin (2)
- vähän haittaa aiheuttaviin bugeihin (1).

Painotettujen bugien keskiarvollinen määrä neljännesvuotta kohden laski 1774:stä 1591:een. Kaikkein vakavimpien, julkaisun estävien bugien, määrä tippui vielä radikaalimmin Scrumin 65:stä Kanbanin 48:aan.

Työn tuottavuus, joka laskettiin kaavalla koodirivimuutosten määrä jaettuna mittausajanjakson kehittäjien määrällä, parani. Tällä menetelmällä laskettuna bugien korjaamisen tuottavuus laski noin 11 %, mutta kehitysjonokohteiden kehityksen tuottavuus nousi 21 %.

Kaiken kaikkiaan tutkimuksessa todetaan, että Scrumin vaihtaminen Kanbaniin puolitti tehtävien läpimenoajan, laski bugien määrää kymmenyksellä ja paransi tuottavuutta selkeästi (Sjøberg ym. 2012).

Kolmannessa esiteltävässä tutkimuksessa seurattiin yhdeksänhenkistä tiimiä, joka työskenteli BBC Worldwide -yrityksessä. Tutkimuksessa haluttiin selvittää, parantaako Lean-ajattelumallin ja Kanbanin käyttö ohjelmistokehitysprosessin suorituskykyä. Seurantajakson alkaessa lokakuussa 2008 tiimin katsottiin olevan jo vakaalla pohjalla. Seurantajakso kesti vuoden 2008 lokakuusta vuoden 2009 lokakuuhun. (Middleton & Joyce 2012.)

Tulokset osoittavat selkeästi, että yli vuoden kestäneen seurantajakson aikana ohjelmistokehitystiimin läpimenoaika parani 37 % mahdollistaen nopeamman työskentelyn, toimitusvarmuus kasvoi 47 % tehden työn suunnittelusta ja aikataulujen arvioinnista helpompaa ja asiakkaiden ilmoittamien virheiden määrä laski 24 % parantaen ohjelmiston laatua. (Middleton & Joyce 2012.)

Tiimi arvioi itse suoriutumistaan ja suurimmiksi syiksi tehokkuuden parantumiseksi olivat:

1. Vain asiakasarvoltaan arvokkainta työtä tehtiin.

2. Tehty työ toimitettiin ja otettiin käyttöön nopeammin, jolloin arvoa saatiin nopeammin toimitettua.

3. Jätteen muodostumisen riski väärinymmärretyistä tai virheellisistä määrittelyistä minimoitiin.

4. Asiakkailta saadun palautteen mukaan he olivat onnellisempia ja suosivat tätä lähestymistapaa.

Tutkimuksen johtopäätöksissä todettiin Lean-menetelmien käyttöönoton parantaneen ohjelmistokehitystiimin suorituskykyä kiistattomasti, joskin haittapuolena mainittiin, ettei se välttämättä sovi hyvin olemassa oleviin yrityksen käytäntöihin. (Middleton & Joyce 2012).

Oulun yliopiston tekemän kirjallisuuskatsauksen 19 eri tutkimuksen yhteenvetotuloksina todettiin Kanbanin lyhentävän ohjelmistotuotteiden toimitusaikaa, parantavan ohjelmistojen laatua, parantavan kommunikointia ja yhteistyötä, nostavan toimintavarmuutta ja vähentävän asiakkaiden raportoimia virheitä (Ahmad ym. 2013).

Samaisessa kirjallisuuskatsauksessa kuitenkin todettiin Kanbanin vaativan muita toimintamalleja tuekseen ja hybridimalleja, kuten Scrumbania suositeltiin käytettäväksi. Myös organisaatiokulttuurin ja -filosofian muutos koettiin vaikeaksi. Lisäksi menetelmien käyttöön kaivattiin lisää erityisosaamista ja koulutusta (Ahmad, Markkula & Oivo 2013).

7 Kanbanin käyttöönotto

Kanbanin käyttöönottoa kokeiltiin projektissa, jonka tavoitteena oli rakentaa tilaajayritys Toinen veli Oy:n KiinteistöVELI-tuotteen ympärille blogisivusto. Projektin katsottiin olevan pieni, mutta antavan hyvän käsityksen Kanbanin käyttöönoton mahdollisesta yksinkertaisuudesta tai monimutkaisuudesta.

Organisaatio ei ollut aiemmin käyttänyt Lean- tai Kanban-menetelmiä, mutta ketteristä ohjelmistokehitysmenetelmistä käytössä oli Scrum. Tiimi oli pieni ja koostui kahdesta yrityksen omistajasta. Asiakkaana toimi yrityksen talousjohtaja. Kanban-taulu päätettiin

järjestää verkko-ohjelmiston avulla asiakkaana toimivan talousjohtajan etäisen sijainnin vuoksi.

Lisäksi projekti aikataulutettiin ja Kanban-taulu ja siihen sisältyvät rajat määriteltiin selkeästi.

7.1 Projektin kuvaus ja tavoitteet

Projektin tavoitteena oli rakentaa insinöörityön tilaajayritys Toinen veli Oy:lle blogisivusto, jolla yritys voi kirjoittaa KiinteistöVELI-tuotteeseensa liittyviä julkaisuja julkisessa verkossa sijaitsevalle blogilleen. Projektin alkuvaatimuksina tiedettiin, että blogin piti olla visuaalisesti miellyttävä ja helppolukuinen, ja sinne piti pystyä kirjoittamaan artikkeleita nopeasti.

Ohjelmistokehityksen kannalta projekti oli hyvin pieni, mutta tavoitteena olikin aikaansaada toimiva Kanban-organisaatio ainakin projektin ajaksi.

Projektin tilaajana ja asiakkaana toimi Toinen veli Oy:n talousjohtaja Valter Pasanen, joka myös esitti Lean-mallin mukaisesti määrittelyjä ohjelmistoon. Hänen oli määrä lisätä aktiivisesti kohteita projektitiimin Kanban-taululle. Ohjelmistokehittäjien tehtävänä oli toteuttaa projekti Leanin ja Kanbanin sääntöihin nojaten.

7.2 Organisaation tausta ja esivaatimukset

Toinen veli Oy:ssä oli aiemmin käytetty tehokkaasti Scrumia joko niin, että asiakkaat olivat olleet mukana kehitysprojektissa koko projektin ajan tai siten, että asiakasta oli simuloitu esimerkiksi sisäisissä kehitysprojekteissa. Tiimi oli tottunut päivämääriin sidottuihin toimitusaikoihin, päiväpalavereihin ja pyrähdysten suunnitteluun. Aiempaa kokemusta tiimillä ei Lean-mallin mukaisesta työskentelystä ollut.

Ajatuksena oli kokeellisesti ajaa läpi yksi Kanban-projekti tilaajayritys Toinen veli Oy:ssä. Taustatutkimuksen perusteella (Leppäniemi 2013) Lean-mallilla toimivalta tiimiltä edellytetään, että tiimin jäsenet ovat vähintään 1) halukkaita jatkuvaan oppimiseen ja 2) vastuunottoykykyisiä. Projektin kehitystiimi koostui kahdesta henkilöstä, minusta itsestäni ja Antti Karkimosta. Tiimi täytti edellä mainitut esivaatimukset hyvin.

Tiimin persoonallisuusvaatimusten lisäksi organisaation on sallittava muutos ja annettava tiimille valta ja vastuu, Fraktio Oy:n toimitusjohtajan Jesse Peuralan sanojen mukaan ”johtajan on oltava enemmänkin valmentaja ja urheilujoukkueesta tuttua mehenkeä pitää saada enemmän”. Tämän projektin tapauksessa johdolta saatavaa valtuutusta ei tässä tapauksessa tarvittu - tiimi koostui tilaajayrityksen kahdesta suurimmasta omistajasta.

Muita esivaatimuksia olivat asiakkaalle avattavat kommunikointiväylät, visuaaliset työnkulun esitystyökalut ja turhan raportoinnin vaatimusten poistaminen (Rubrich 2010). Projektin ajan asiakkaamme sijaitsi Saksassa, joten kommunikointiväyläksi sovittiin ensisijaisesti Skype ja toissijaiseksi matkapuhelin. Visuaaliset työnkulun esitysvälineet esitellään seuraavassa kappaleessa. Organisaatio oli rakenteeltaan kevyt, eikä olemassa olevaa raportointia ollut juurikaan, joten nämä aiemmin mainitut turhan raportoinnin vaatimusten poistaminen oli jo projektin alkaessa tehty.

Näin ollen kaikki esivaatimustiedot olivat selvillä ja täytetty.

7.3 Työvälineet

Tiimille helpoin ja miellyttävin tapa olisi ollut käyttää tilaajayrityksen toimistolla sijaitsevaa fyysistä valkotaulua, jolle tehtävät lisättäisiin muistilappujen avulla. Asiakkaamme etäisen sijainnin vuoksi (ks. luku 7.2) päätimme, että tarvitsemme jonkin verkossa toimivan ohjelmiston, joka on helppokäyttöinen ja monen henkilön käytettävissä fyysisesti eri sijainneista.

Mahdollisia ohjelmistoja käytiin läpi muutamia, mutta työssä päädyttiin lopulta käyttämään Kanbanize-tuotetta. Se on perusominaisuuksiltaan maksuton työkalu, joka skaalautuu hyvin organisaation tarpeiden mahdollisesti kasvaessa.

Versionhallinta päätettiin hoitaa Git-versionhallintaohjelmistolla ja varsinainen ohjelmointityö jokaisen henkilön itse valitsemalla ohjelmointiympäristöllä.

7.4 Projektin aikataulutus

Kuten Lean-malliin kuului, varsinaisia aikatauluja ei asetettu. Sellaiset rajat kuitenkin sovittiin, että projekti alkaisi 17.03.2014 ja valmistuisi, kun valmistuisi. Projektin yksinkertaisen luonteen vuoksi sen oletettiin valmistuvan nopeasti. Kanban oli kuitenkin tiimille uusi ja sen käytön opettelun katsottiin vievän jonkin verran aikaa.

7.5 Tehtävien suunnittelu

Projektin käynnistysvaiheessa valittiin sarakkeet ja asetettiin työmääräraja luvun 4.2 mukaisesti. Tässä projektissa päätettiin, että sarakkeita on 1) tehtävälista, 2) tekeillä, 3) testauksessa ja 4) tehty.

Taulukko 1. Kanban-taulun sarakkeet ja työmäärärajat.

	Tehtävälista	**Tekeillä**	**Testauksessa**	**Tehty**
Sarakkeen kuvaus	Tähän sarakkeeseen tulevat kaikki ne työt, jotka tiimi ottaa asiakkaan hallinnoimasta kehitysjonosta.	Tähän sarakkeeseen otetaan tehtävälistalta seuraava työ, kun sitä aletaan tekemään.	Tähän otetaan tekeillä ollut työ sen valmistuttua.	Kun testaus on tehty, voidaan tehtävä lopulta tuoda tänne.
Työmääräraja	5	2	2	-

Tehtävälistalle voitiin ottaa enintään viisi tehtävää kerrallaan. Tätä rajoitettiin siksi, ettei sinne lisättäisi kerralla kaikkia mahdollisia tehtäviä ja että taulu pysyisi helposti hahmotettavana. Päätettiin olla käyttämättä pariohjelmointina ja ohjelmointityöhön oli käytettävissä kaksi ihmistä. Niinpä päätettiin asettaa tekeillä oleviin töihin ja testauksessa oleviin töihin työmäärärajaksi kaksi. Silloin kummallakin tiimin jäsenellä saattoi olla samaan aikaan työn alla oleva tehtävä, mutta ei yhtään enempää. Viimeiselle sarakkeelle ei luonnollisestikaan tarvittu työmäärärajaa, koska siellä ei enää toteutettu mitään.

Tehtävien kokoon ei lähteistä löytynyt suoraa vastausta, mutta pienet tehtävät mahdollistivat paremman seurattavuuden. Nyrkkisäännöksi otettiinkin, että yksi tehtävä ei saisi kestää yli yhtä työpäivää yhdeltä tiimin jäseneltä. Kaikki tehtävät pyrittiin pilkkomaan pieniksi mutta loogisiksi osakokonaisuuksiksi.

8 Käyttöönoton tulokset

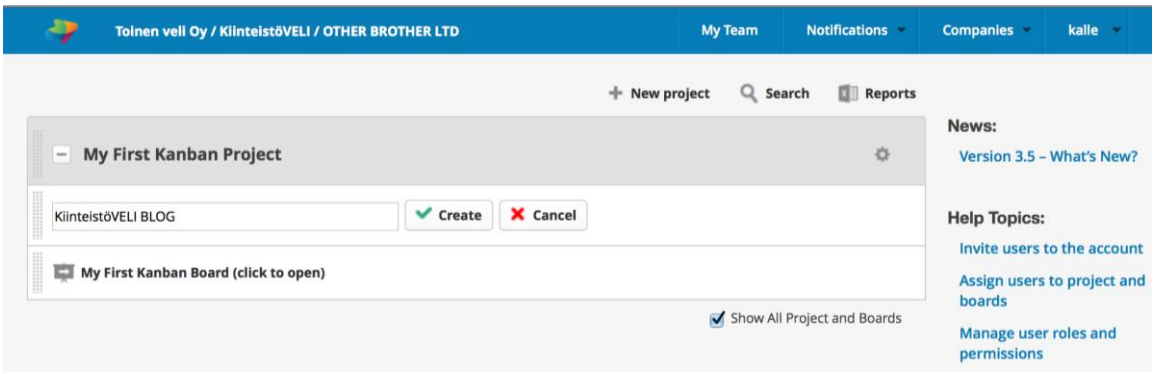
Projekti käynnistyi aikataulun mukaisesti ja asiakkaalle annettiin vapaat kädet ideoida tuotteen kehitysjonoon kehityskohteita. Alkuvaiheen haasteiden jälkeen asiakkaalta tulevat vaatimukset olivat hyvin muotoiltuja, joskin kehitystiimi päätti pilkkoa vaatimukset pienempiin tehtäviin.

Kanban- ja Lean-menetelmien opettelu vei jonkin verran aikaa, vaikka ne koettiinkin helposti sisäistettäviksi. Suurin haaste oli asiakkaan sitouttaminen projektiin, joka lopulta onnistui hyvin. Vaikeaksi asiakasyhteistyön kannalta osoittautui myös työskentely ilman iteraatioita ja selkeitä aikarajoja.

Kaiken kaikkiaan käyttöönottoprojekti sujui hyvin. Tiimi koki työnkulun visualisoinnin positiiviseksi ja helpottavan projektin tilan hahmottamista. Sarakekohtainen työmäärä-raja antoi yhdelle asialle kerrallaan työrauhan.

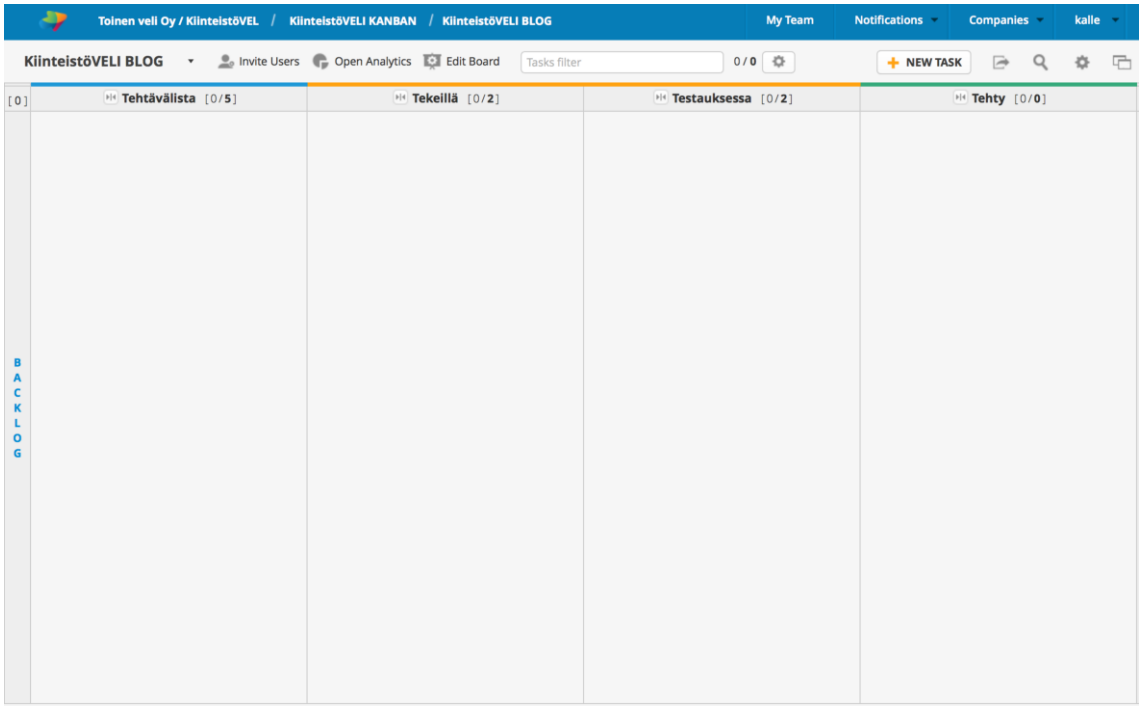
8.1 Aloituspalaveri ja ohjeistus

Projekti käynnistettiin aikataulun mukaisesti, jolloin myös käytiin läpi käyttämämme Kanbanize-järjestelmä ja kerrottiin lyhyesti, miten projektin on tarkoitus edetä. Tässä vaiheessa myös luotiin varsinainen Kanban-taulu Kanbanize-järjestelmään. Järjestelmässä voisi olla useampia tauluja, mutta projektiin tarvittiin vain yksi.



Kuva 6. Kuvankaappaus Kanban-taulun luonti-ikkunasta Kanbanize-järjestelmässä (Lähde: Kanbanize.com)

Kun kuvan 6 taulu oli luotu, asetettiin Kanbanizessa luvussa 7.5 määrittelemämme sarakkeet ja esitetyt työmäärärajat. Sarakkeita olivat tehtävälista, tekeillä, testauksessa ja tehty.



Kuva 7. Kuvankaappaus Kanbanize-järjestelmässä luodusta Kanban-taulusta (Lähde: Kanbanize.com)

Kuvasta 7 havaitaan, että Kanbanize-järjestelmän Kanban-taulu muistuttaa visuaalisesti luvussa 4.2 esiteltyä Kanban-taulua. Kun järjestelmiin luotiin tehtäviä, ne muistuttivat muistilappuja ja olivat helposti hahmoteltavissa.

Asiakkaan kanssa sovittiin hänen saavan visioida vapaasti tuotteen kehitysjonoon asioita, joista pitäisi käydä mahdollisimman hyvin selväksi, mitä hän haluaa. Kehitystiimi otti tehtävälistalle WIP-limitin mukaan kerrallaan enintään viisi tehtävää kehitysjonosta, asiakkaan asettamassa tärkeysjärjestyksessä.

8.2 Projektin eteneminen

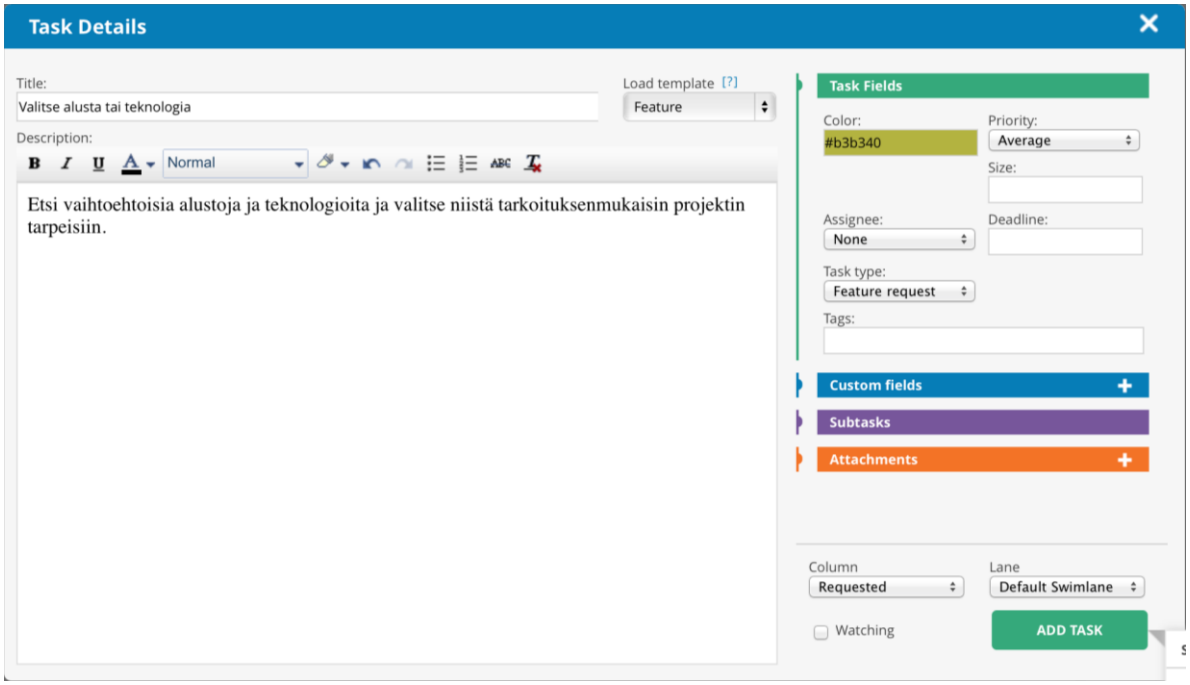
Asiakkaalle annettiin vapaat kädet ideoida tuotteen kehitysjonoon kehityskohteita, mutta varsin pian havaittiin tehtävien olevan hyvin isoja kokonaisuuksia, kuten ”Tee blogi”. Asiakasta pyydettiin kirjoittamaan kehitysjonon kohdat pikemminkin käyttäjätarinamuo-

toon (engl. *User Story*), jossa ajatus on ilmaista kehitettävä ominaisuus jonkin ohjelmiston, eli tässä tapauksessa blogin, käyttäjän näkökulmasta. Esimerkiksi näin:

> ”Lukijana haluan nähdä uusimmat blogikirjoitukset ensin ja pystyä selaamaan niitä tarvittaessa eteen- ja taaksepäin”.

Uuden ohjeistuksen ansiosta asiakas alkoi tuottamaan parempia kehitysjonokohteita, mutta edelleen jotkin kokonaisuudet olivat liian isoja ollakseen yksittäisiä tehtäviä. Siksi päätettiin, että kehitystiimi voi tarvittaessa pilkkoa yhden käyttäjätarinan pienempiin osiin, jotta projektissa pystytään pitämään kiinni otsikossa 7.5 määritellystä tavoitteesta, jossa yksi tehtävä ei kestäisi yhtä työpäivää pidempään.

Kanbanize tarjoaa paljon mahdollisia Kanbaniin liittymättömiäkin ominaisuuksia tehtävän luontivaiheessa. Näitä ovat esimerkiksi suoraan vastuuhenkilön nimeäminen tai tietyt ylimääräiset tunnisteet. Tiedot saattoi jättää tyhjäksi, jolloin verkossa sijaitsevan Kanban-taulun päivittäminen oli helppoa ja nopeaa.



Kuva 8. Esimerkkikuva Kanbanize-työkalun tehtävän lisäysvaiheesta (Lähde: Kanbanize.com)

Kuvasta 8 nähdään, että tehtävälle voidaan määrittää prioriteetti, koko, valmistumisen takaraja ja eri värejä. Lisäksi se voitaisiin suoraan osoittaa jonkun henkilön tehtäväksi, mutta silloin kyse ei olisi JIT:n mukaisesta työskentelystä. JIT ja siihen liittyvä pull-

mekanismi on esitelty luvussa 4.2. Kanbanizen tarjoamista ominaisuuksista käytettiin erilaisia värejä ja prioriteettia, sekä luonnollisesti tehtävän otsikko- ja kuvauskenttiä.

8.3 Projektin valmistuminen ja johtopäätökset

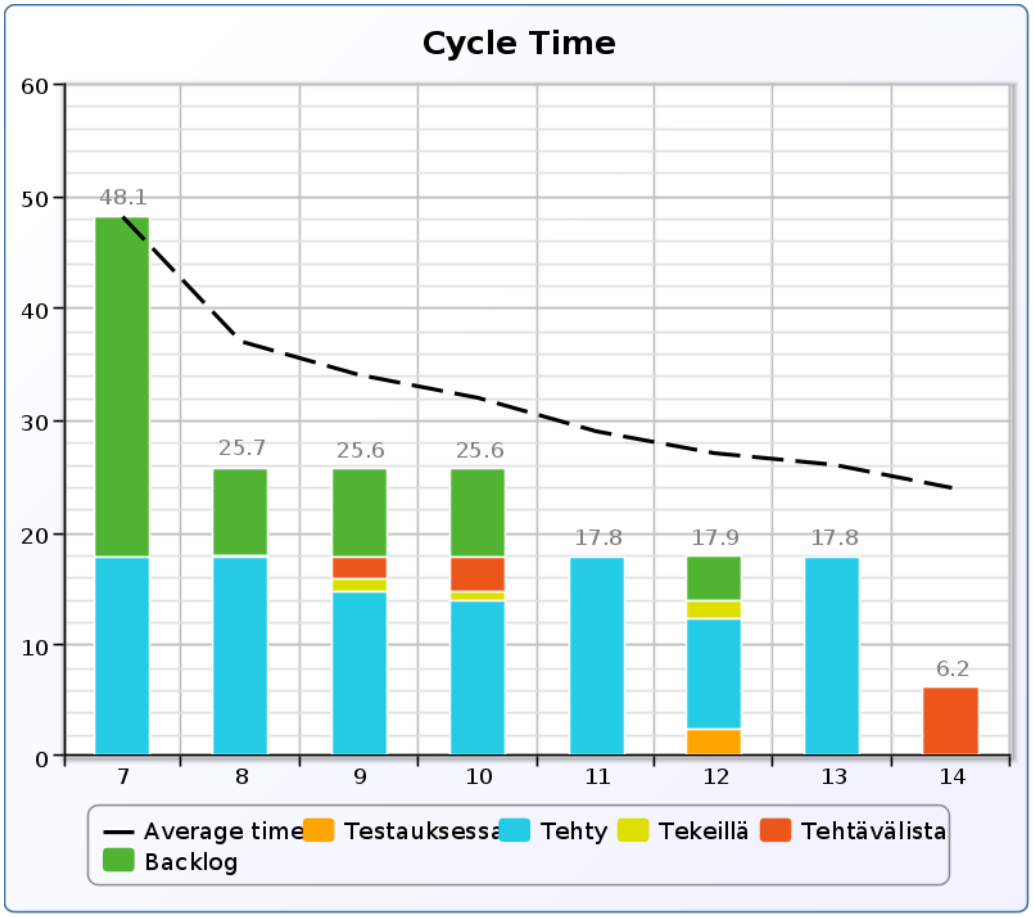
Projekti päättyi 18.4.2014 kaikkien tehtävien ollessa tehtyjä. Asiakas totesi työn olevan valmis ja kaikkien hänen asettamiensa tavoitteiden mukainen. Lopputuotoksena syntyi blogi, joka on artikkelien kirjoittajalle helppokäyttöinen ja lukijalle mukavalukuinen.

Kuva 9. Kuvankaappaus projektissa valmistuneesta blogisivustosta (Lähde: Toinen veli Oy)

Kuvassa 9 on kuvankaappaus syntyneestä blogin etusivusta. Kirjoitusta klikkaamalla pääsi lukemaan artikkelin kokonaan. Varsinaisella artikkelisivuilla pystyi sekä lukemaan että kirjoittamaan kommentteja.

Vaikka projektiin kului kokonaisuudessaan aikaa noin kuukausi, niin todellisuudessa tiimi ei pystynyt tekemään pelkästään kyseistä projektia. Projektin aikana kertyikin todellisia kirjattuja työtunteja vain 31,5. Jos tiimi olisi keskittynyt pelkästään tähän projektiin ja asiakas olisi pystynyt tuottamaan kehityskohteita riittävän tiheällä tahdilla, se olisi ollut valmis alle viikossa.

Siitä huolimatta tiimi paransi jatkuvasti työskentelytehoaan, kuten Kanbanissa oletetaankin tapahtuvan ja läpimenoaika laski loppua kohti. Osasyy tähän saattoi olla myös se, että tehtävien luonne myös helpottui loppua kohden. Alkuvaiheessa ulkoasuun liittyvät seikat veivät aikaa enemmän kuin esimerkiksi navigaation implementointi, johon löytyy valmiiksi hyvät ohjelmistokirjastot.



Kuva 10. Kehitysjonokohteiden läpimenoaika laski projektin edetessä (Lähde: Kanbanize.com).

Kuten kuvasta 10 huomataan, läpimenoaika oli useita päiviä, vaikka tosiasiallinen toteutus kesti huomattavan paljon vähemmän. Kuvassa läpimenoajan keskiarvoa kuvaa musta katkoviiva. Tehtävälistalle jäi blogin julkaiseminen, joka kuitenkin odottaa varsinaisen sisällön tuottamista.

Suurimmaksi haasteeksi projektissa osoittautui asiakkaan roolin kirkastaminen ja sujuvan asiakasyhteistyön saavuttaminen. Vaikka asiakkaaseen oli olemassa oleva luottamussuhde, ja kommunikointi onnistui hyvin, oli aika-ajoin haasteellista saada asiakkaalta riittävän nopeasti uusia kohteita.

Toinen vaikeasti sisäistettävä asia aiemmin aikarajatussa ympäristössä toimineelle kehitystiimille oli itse ymmärtää työskentelevänsä ilman varsinaisia päivämäärärajoituksia. Vielä suurempi haaste oli oppia viestimään tätä asiakkaalle: enää ei voinut sanoa, mitä on valmiina tiettyyn päivämäärään mennessä, vaan asiakkaan piti sitoutua jatkuvasti olemaan mukana kehitysprosessissa ja sitä kautta saada vakuuttumaan siitä, että tiimi tekee töitä asiakkaan priorisointilistan mukaisessa järjestyksessä.

Silti projektin aloittaminen oli helppoa, ja kehitystiimi koki työmäärärajat ja työnkulun visualisoinnin positiivisena asiana. Työmääräraja estää säntäilemästä tekemään uusia tehtäviä ennen kuin vanha on valmistunut. Visuaalinen taulu, jopa verkossa esitettynä, on helpommin hahmoteltavissa kuin pelkästään tekstimuotoinen lista asioista, ja siitä näkee nopeasti, mitä on vielä tekemättä.

Uskon, että jos kehitystiimi on riittävän pieni, voidaan mihin tahansa organisaatioon tuoda Kanban mukaan vain kohtalaisella vaivalla, mikäli sen taustalla olevat ajatukset ymmärretään ja niille annetaan organisaatiossa tilaa. Juuri käytön opettelun helppous on myös suurin yksittäinen syy sille, että Software Factory käyttää Kanbania tai siitä sovellettua Scrumbania (Fagerholm 2013).

Tiimijohtamisen asiantuntija J. Richard Hackmanin nyrkkisäännön mukaan tiimissä saisi olla enintään yhdeksän ihmistä (Leppänen 2010), mutta itse mieltäisin Kanbanin soveltuvan helposti tiimiin, joka sisältää enintään kuusi jäsentä. Syy tälle kokorajoitukselle on se, että jos tiimi on suurempi, niin Kanban-taulun koko kasvaa liikaa. Luvussa 4.2 esitetty peruspilari on se, että taulun on oltava kerralla nähtävissä.

Vaikka projekti sujui hyvin, ja se hoidettiin tehokkaasti, ei Toinen veli Oy kuitenkaan ole valmis siirtymään heti tämän pilottiprojektin jälkeen käyttämään ainoastaan Kanbania. Siitä huolimatta se päätettiin ottaa lähempään tarkasteluun ja kokeilla sitä myös sellaisen asiakkaan kanssa, johon on jo luotu syvä luottamussuhde ja jolla on riittävä teknistä kyky osallistua projektiin niin täysipainoisesti kuin se on tarpeen.

9 Johtopäätökset

Kanban ja Lean-ajattelumalli ovat hyvin uusia käsitteitä ohjelmistokehitystyössä ja kuten Oulun yliopiston kirjallisuuskatsauksessa todetaan, tieteellistä tutkimustietoa Kanbanin käytöstä ohjelmistokehitystyössä on selvästi liian vähän. Lisäksi Kanban määritellään kirjallisuudesta hyvin abstraktilla tasolla, eikä sellaista tutkimusta, jossa Kanban-prosessimallia selvästi ja syvällisesti esitettäisiin ole olemassa (Ahmad ym. 2013).

Siitä huolimatta on olemassa rohkaisevia esimerkkejä Kanbanin ja Leanin onnistuneesta käyttöönotosta useissa eri organisaatioissa. Tulokset ovat olleet hyviä, mutta tutkimusten seuranta-ajat eivät ole olleet erityisen pitkäkestoisia. (Sjøberg ym. 2012 ja Middleton & Joyce 2012.)

Voidaankin kysyä, johtuvatko Kanbanin käyttöönoton positiiviset tulokset vain tiimin positiivisesta ja ennakkoluulottomasta suhtautumisesta johonkin uuteen, vai pystytäänkö Kanbanin ja Leanin avulla jatkuvasti tehostamaan toimintaa turhien töiden poistamisen avulla. Tämä kysymys jää odottamaan pidempiaikaisen tutkimuksen valmistumista.

Tämän insinöörityön käytännön osuuden perusteella Kanban entisestään virtaviivaistaa prosessia ja pyrkii johdonmukaisesti poistamaan sellaisia tehtäviä, jotka eivät tuota arvoa asiakkaalle. Se pyrkii keskittymään absoluuttisesti sellaisiin asioihin, jotka ovat relevantteja asiakkaalle, lisäävät ohjelmiston laadukkuutta ja tehostavat ohjelmistokehitysprosessia. Tämä virtaviivaistaminen ja turhien käytäntöjen poistaminen onkin suurin hyöty, mitä Lean ja Kanban voivat ohjelmistokehitystyöhön tarjota.

Poppendieckin esittämät yleiset periaatteet (ks. luku 3.3) sopivat hyvin myös muihin ketteriin ohjelmistokehitysmalleihin ja vaikka Lean-ajattelumalli ja Kanban-prosessimalli todistetusti tehostavat ohjelmistokehitysprosessia, eivät ne itsessään ole oikotie onneen.

Työn tavoitteena oli selvittää, onko Lean-ajattelumallin mukaisten prosessimallien, kuten Kanbanin, käyttäminen järkevää ohjelmistokehitysprojekteissa ja jos, niin minkälaisissa. Lean-ajattelumalliin pohjautuvan Kanbanin käyttäminen on järkevää sellaisissa ohjelmistokehitysprojekteissa, joita tehdään sellaiselle asiakkaalle, jolla on tekninen kyky osallistua aktiivisesti projektin määrittelyyn ja johon joko on olemassa tai vaihtoehtoisesti on synnytettävissä vahva luottamussuhde.

10 Yhteenveto

Insinöörityössä selvitettiin Lean-ajattelumallin ja Kanban-prosessimallin käytön soveltuvuutta ohjelmistokehitysprojekteissa. Lisäksi tavoitteena oli tutkia näiden mallien hyötyjä ja haittoja sekä verrata niitä muihin ketteriin ohjelmistokehitysmenetelmiin, ensisijaisesti Scrumiin.

Työn tavoitteet saavutettiin hyvin, vaikka tieteellistä tutkimustietoa aiheesta oli saatavilla vain niukasti. Saavutetut kirjalliset tulokset ja käytännön projektiosuus osoittivat, että Lean ja Kanban tehostavat ohjelmistokehitysprosessia nopeuttamalla läpimenoaikaa, lisäämällä toimitusvarmuutta ja vähentämällä virheitä ohjelmistossa. Näin ollen menetelmät soveltuvat ohjelmistokehitysprojekteihin hyvin.

Leanin ja Kanbanin suurimpana haittana koettiin kyvyttömyys ennakoida tarkkaan välietappeja tai määrittää etukäteen aikatauluja projekteille. Tämä voidaan kokea ongelmaksi, jos asiakkaan kanssa ei ole hyvää ja molemminpuolista luottamussuhdetta. Menetelmän tuottamat hyödyt kuitenkin ovat suurempia kuin siitä koituvat haitat.

Tätä työtä tehdessä oli haasteellista ymmärtää eroja muiden ketterien ohjelmistokehitysmenetelmien ja Leanin ajattelumallien välillä. Tämän työn puitteissa pyrittiin myös avaamaan Leanin ja Kanbanin ympärillä leijailevaa ymmärtämättömyyttä ja siinäkin onnistuttiin. Tulevia töitä ajatellen Leanin ja Kanbanin tarkempi ja käytännönläheisempi kuvaaminen ohjelmistokehitystyössä olisi tarpeen, jotta sen käyttöönotto voisi olla ohjelmistokehitysorganisaatiossa helpompaa.

Lähteet

Abrahamsson Pekka, Salo Outi, Ronkainen Jussi, Warsta Juhani. 2002. Agile software development methods. Review and analysis. Oulu: VTT.

Agile software development. 2013. Verkkodokumentti. <http://en.wikipedia.org/wiki/Agile_software_development>. 23.07.2013. Luettu 26.07.2013.

Ahmad, Markkula, Ovio. 2013. Kanban in software development. Oulu: Oulun yliopisto.

Beck, Beedle, van Bennekum, Cockburn, Cunningham, Fowler, Grenning, Highsmith, Hunt, Jeffries, Kern, Marick, Martin, Mallor, Shwaber, Sutherland. 2001. Agile Manifesto. Verkkodokumentti. <http://agilemanifesto.org>. Luettu 13.04.2014.

Begel, Andrew & Nagappan Nachiappan. 2007. Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study. Redmod: IEEE Computer Society.

Being an Effective Product Owner. Verkkodokumentti. <http://www.scrumalliance.org/community/articles/2007/april/being-an-effective-product-owner>. 23.04.2007. Luettu 05.11.2013.

Burlingham & Gendron. 1989. The Entrepreneur of the Decade - An interview with Steven Jobs, Inc's Entrepreneur of the Decade.

Bye, Kent. 2013. Verkkodokumentti. Myths of Kanban. <http://kentbye.com/post/42447226722/myths-of-kanban>. 06.02.2013. Luettu 24.03.2014.

Definition of iterative. Verkkodokumentti. <http://www.thefreedictionary.com/iterative>. Luettu 26.07.2013.

Dyba Tore, Dingsøyr Torgeir. 2008. Empirical studies of agile software development: A systematic review. Trondheim: Elsevier.

Empiirinen tutkimus. 2013. Verkkodokumentti. <http://fi.wikipedia.org/wiki/Empiirinen_tutkimus>. 15.03.2013. Luettu 09.08.2013.

Fagerholm, Fabian. 2013. Tohtorikoulutettava, Helsingin yliopisto. Helsinki. Haastattelu 02.10.2013.

Fichtner, Abby. 2012. Agile Vs. Lean: Yeah, yeah, What's the Difference?. Verkkodokumentti. <http://www.hackerchick.com/2012/01/agile-vs-lean-yeah-yeah-whats-the-difference.html>. 01.01.2012. Luettu 13.04.2014.

Fast Facts about Microsoft. Verkkodokumentti. <http://www.microsoft.com/en-us/news/inside_ms.aspx>. Luettu 18.10.2013.

Gulaz, Aycan. Verkkodokumentti. Finding the right task size in kanban. <http://flow.io/finding-the-right-task-size-in-kanban.html>. Luettu 24.03.2014.

Ikonen, Marko. 2011. Lean Thinking in Software Development: Impacts of Kanban on Projects. Helsinki: Unigrafia.

Iterative and Incremental Development. 2013. Verkkodokumentti. <http://en.wikipedia.org/wiki/Iterative_and_incremental_development>. 21.07.2013. Luettu 26.07.2013.

Jaakonaho, Kaisa. 2009. Sopimusmallit ketterän lähestymistavan mukaisissa projekteissa. Jyväskylä: Jyväskylän yliopisto.

Jousi, Outi. 2011. Verkkoesitys. Ketterät ohjelmistokehitysmenetelmät julkisissa hankinnoissa. <http://www.slideshare.net/codento/agile-aamiaisseminaari-codento23112011final>. 23.11.2011. Katsottu 26.07.2013.

Just-in-Time. 2013. Verkkodokumentti. <http://fi.wikipedia.org/wiki/Just-In-Time>. 31.12.2013. Luettu 26.02.2014.

Kanban - Creating a Kaizen Culture and evolving Lean Software Engineering Solutions. Verkkoesitys. <http://www.slideshare.net/deimos/david-anderson-kanban-at-q-con>. 16.03.2008. Katsottu 16.11.2013.

Kanban Basics. Verkkoesitys. <http://www.slideshare.net/jprokop1/kanban-basics-1229209>. 31.03.2009. Katsottu 06.10.2013.

Kanban Tool. 2013. Verkkodokumentti. Kanbantool. <http://kanbantool.com>. Luettu 14.06.2013.

Kanban-menetelmä. 2011. Verkkodokumentti <http://reaktor.fi/osaaminen/kanban/>. 01.07.2011. Luettu 15.05.2013.

Kanban: succesful evolutionary change for your technology business – David J. Anderson. Verkkovideo. <https://www.youtube.com/watch?v=hpONxfefERA>. 25.03.2013. Katsottu 16.11.2013.

Kanban. 2013. Verkkodokumentti. <http://en.wikipedia.org/wiki/Kanban_(development)>. 05.06.2013. Luettu 07.06.2013.

Kanban. 2013. Verkkodokumentti. <http://fi.wikipedia.org/wiki/Kanban>. 04.06.2013. Luettu 07.06.2013.

Karhatsu & Moilanen. 2009. Mura, muri, muda – lean-filosofian hukkakäsitteet ohjelmistokehityksessä. Seminaarityö. Helsinki: Helsingin yliopisto.

Kvantitatiivinen tutkimus. 2013. Verkkodokumentti. <http://fi.wikipedia.org/wiki/Kvantitatiivinen_tutkimus>. 15.03.2013. Luettu 09.08.2013.

Larman & Basili. 2003. Lehtiartikkeli. Interative and incremental developments. a brief history. IEEE Computer Society.

Lekman, Lare. 2009. Verkkodokumentti. Mikä ihmeen Kanban?. <http://larelekman.com/2009/09/26/mika-ihmeen-kanban/>. 26.09.2009. Luettu 18.05.2013.

Leppänen, Jorma. 2010. Lehtiartikkeli. Tiimi tarvitsee kyseenalaistajan. Prima 8/2010.

Leppäniemi, Panu. 2013. Ohjelmistokehittäjä, Fraktio Oy, Helsinki. Haastattelu 14.06.2013.

Middleton, Peter & Joyce, David. 2012. Lean Software Management: BBC Worldwide Case Study. IEEE Transactions on Engineering Management, Vol. 59, NO 1.

Mitchell, Ian. 2013. Verkkodokumentti. Why Stretched Teams do ScrumBan. <http://agile.dzone.com/articles/why-stretched-teams-do>. Luettu 29.06.2013.

Murphy Brendan, Bird Christian, Zimmermann Thomas, Williams Laurie, Nagappan Nachiappan, Begel Andrew. 2013. Have Agile Techniques been the Silver Bullet for Software Development at Microsoft? Microsoft.

Murray, Martin. Verkkodokumentti. Introduction to Kanban. <http://logistics.about.com/od/tacticalsupplychain/a/Introduction-To-Kanban.htm>. Luettu 14.06.2013.

NATHAN-REGIS, Bodje. 2012. Evaluation of the most used Agile methods (XP, LEAN, Scrum). Bangalore: Christ University.

Nygrén, Jaakko. 2012. Tuoteomistajan rooli ketterässä ohjelmistokehityksessä. Helsinki: Helsingin yliopisto.

Oppiminen ja innovaatio työelämässä. Verkkovideo. <http://www.youtube.com/watch?v=1Da1PSPWu0U>. Katsottu 05.10.2013.

Pahuja, Savita. 2012. What is Scrumban?. Verkkodokumentti. <http://www.solutionsiq.com/resources/agileiq-blog/bid/87799/What-is-Scrumban>. 08.04.2012. Luettu 29.06.2013.

Petersen, Kai & Wohlin, Claes. 2009. Measuring the Flow in Lean Software Development. Ronneby: Blekinge Institute of Technology.

Peurala, Jesse. 2013. Toimitusjohtaja, Fraktio Oy, Helsinki. Haastattelu 14.06.2013.

Poppendieck & Poppendieck. 2003. Lean Software Development: An Agile Toolkit. 1. painos. Boston: Addison Wesley.

Poppendieck & Poppendieck. 2007. Implementing Lean Software Development. 9. painos. Boston: Pearson Education (US).

Poppendieck, Mary & Cusumano, Michael A. 2012. Lehtiartikkeli. Lean Software Development: A Tutorial. USA: IEEE Computer Society.

Project Google – Get found by Google. Verkkodokumentti. <http://markhoogveld.wordpress.com/tag/scrum-overview/>. 11.03.2013. Luettu 06.11.2013.

Push-pull Strategy. Verkkodokumentti. <http://en.wikipedia.org/wiki/Push–pull_strategy>. 04.11.2013. Luettu 10.11.2013.

Rinko-Gay, Bill. 2013. Verkkodokumentti. You May Be A Scrum-But. <http://www.scrumalliance.org/community/articles/2013/february/you-may-be-a-scrum-but>. Luettu 29.06.2013.

Rubrich, Larry. 2010. 14 things to keep in mind when implementing kanbans. Verkkodokumentti. <http://www.reliableplant.com/Read/22368/implementing-kanbans-lean>. 27.01.2010. Luettu 01.03.2014.

Sandvik, Risto. 2011. Verkkodokumentti. Ketterä ohjelmistokehitys ja toimitussopimukset. <http://www.twobirds.com/Finnish/News/Articles/Sivut/Ketterä_ohjelmistokehitys_ja_toimitussopimukset.Aspx>. 26.06.2011. Luettu 26.07.2013.

Scrum Guiden 2013 muutokset. Verkkodokumentti. < http://larelekman.com/2013/09/07/scrum-guiden-2013-muutokset/>. 07.09.2013. Luettu 04.01.2014.

Scrum. 2013. Verkkodokumentti. <http://fi.wikipedia.org/wiki/Scrum>. 16.04.2013. Luettu 09.08.2013.

ScrumBut. Verkkovideo. <https://www.youtube.com/watch?v=tgBkvS-q_fA>. Katsottu 29.06.2013.

Sjøberg Dag, Johnsen Anders, Solberg Jørgen. 2012. Quantifying the Effect of Using Kanban versus Scrum: A Case Study. Oslo: IEEE Computer Society.

Srinivasan, Jayakanth & Lundqvist, Kristina. 2009. Case Study. Using Agile Methods in Software Product Development: A Case Study. MŠlardalen University: IEEE Computer Society.

Steve Jobs on Apple's Resurgance: "Not a one-man show". Verkkodokumentti. <http://www.businessweek.com/bwdaily/dnflash/may1998/nf80512d.htm>. 12.05.1998. Luettu 26.02.2014.

Survey Confirms Scaling Agile Across The Organisation Is Still A Challenge. Verkkodokumentti. <http://www.infoq.com/news/2012/05/survey-agile-scaling>. 19.05.2013. Luettu 18.10.2013.

Takeuchi Hirotaka & Nonaka Ikujiro. 1986. Lehtiartikkeli. The New New Product Development Game. Harvard Business Review.

Tauriainen Simo. 2005. Ohjelmistotestauksen kehittäminen. Kajaani: Kajaanin ammattikorkeakoulu.

The Kanban Story: Kanban Board. Verkkodokumentti. <http://brodzinski.com/2009/11/kanban-story-kanban-board.html>. 23.10.2009. Luettu 09.11.2013.

Vesiputousmalli. Verkkodokumentti. <http://fi.wikipedia.org/wiki/Vesiputousmalli>. 31.07.2013. Luettu 07.08.2013.

What is Lean?. Verkkodokumentti. <http://www.lean.org/whatslean/>. Luettu 29.06.2013.

Yeret, Yuval. 2010. Verkkoesitys. Scrumban - Taking Scrum outside its comfort zone. <http://www.slideshare.net/yyeret/scrumban-taking-scrum-outside-its-comfort-zone>. Katsottu 29.06.2013.

You May Be a Scrum-But. 2013. Verkkodokumentti. < http://www.scrumalliance.org/community/articles/2013/february/you-may-be-a-scrum-but>. 01.02.2013. Luettu 27.02.2014.